Investigation of Thermal Electric Cooler-less Microbolometer Infrared Imaging

by

Gregory M. Mahowald

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical [Computer] Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 5, 2003

Author 5/05/03
Department of Electrical Engineering and Computer Science
May 5, 2003

Certified by 05/05/03
Brian Backer
VI-A Company Thesis Supervisor

Certified by 5/20/03
Qing Hu
M.I.T. Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Investigation of Thermal Electric Cooler-less Microbolometer Infrared Imaging

by

Gregory M. Mahowald

Submitted to the

Department of Electrical Engineering and Computer Science

May 5, 2003

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Computer [Electrical] Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The current IR imaging modules developed by BAE SYSTEMS North America use a Thermal Electric Cooler (TEC) to stabilize the Focal Plane Array (FPA). An FPA is an array of microbolometers which are small vanadium oxide elements whose resistances are very sensitive to IR radiation. The ability to run an imaging module without the TEC results in significant reduction in FPA package size, power consumption and module costs. In order to run a module in such a way several variables have to be accounted for. Running the module in TEC-Less mode requires changing the bolometer current as well as adjusting corrective maps in accordance with changes in ambient temperature. This investigation looked at these adjustments in order to determine if TEC-Less imaging was viable for the modules developed at BAE systems. A spatial noise test is used to determine if the TEC-Less images are as good of quality as those taken under normal operation. The results show that TEC-Less imaging over large temperature ranges is possible, and with the adjustments developed in this report, produces images with only slightly higher spatial noise.

Keywords: IR imaging, TEC-Less, microbolometer

Thesis Supervisor: Qing Hu

Title: Professor, Department of Electrical Engineering and Computer Science M.I.T.

Table of Contents

LIST OF FIGURES

List of Tables

Introduction

This thesis was completed at BAE SYSTEMS in Lexington, MA as part of the MIT VI-A Program. The goal was to determine if the imaging modules developed by BAE can produce acceptable IR images without thermal stabilization. The current imaging modules are stabilized by a Thermal Electric Cooler (TEC) which keeps the Focal Plane Array (FPA) at a constant temperature. This research is meant to determine the feasibility of operating these modules without a TEC, known from here on as TEC-Less operation. Removing the TEC provides significant reduction in FPA package size, power consumption and module costs.

Many different tests were conducted involving both hardware and software changes. All the tests were conducted at the same bench setup with the main components being a Standard Imaging Module (SIM), which includes all the circuits and components needed to image IR, a PC, a thermal chamber and a thermal Black Body (BB). Most of the software changes were simulated on the PC in the C++ programming language and then downloaded to the module. Transferring large amounts of data from the PC to the module took a significant amount of time, which contributed to some problems in the tests. Another issue was that only one FPA and module was used for the tests due to problems integrating the large package size of the LAM2D FPA with the FFE designed to hold the smaller CASPER FPA.

The results show that TEC-Less operation is possible, but extensive embedded (in-module) hardware and software changes would need to be implemented. By adjusting offsets and biases with hardware modifications and updating offset maps with software, promising results were obtained over a wide range of temperatures under TEC-Less operation. The images were taken under controlled conditions and were slightly noisier than with the TEC running. Continued investigation is clearly needed with more FPAs and additional correction algorithms being explored. Nevertheless, these results show that TEC-Less operation is possible.

Infrared Radiation

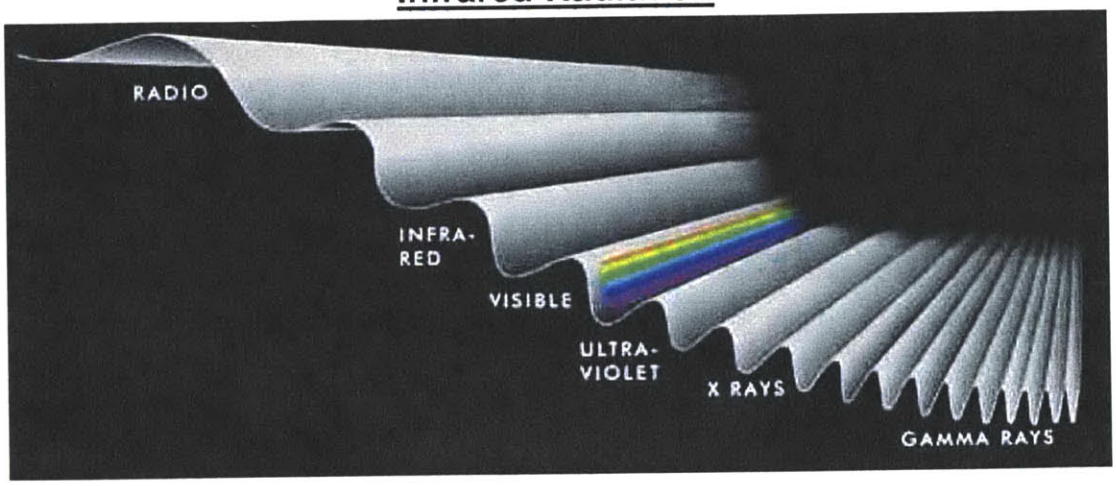


Figure 1: Electromagnetic spectrum

Infrared radiation has a slightly longer wavelength than visible light. Its place in the electromagnetic spectrum can be seen in Figure 1 above. IR radiation, because of its longer wavelength, can penetrate media such as clouds and smoke that visible light cannot. Therefore imaging systems that are very sensitive to IR radiation can be extremely useful in firefighting, astronomy and many military applications.

There are several ways to capture an IR image. One technique is to cool a sensor to very low temperatures using liquid nitrogen. These sensors then directly measure the amount of IR energy coming from the scene. The systems used for this investigation do not use liquid nitrogen and normally run with the TEC at room temperature and are therefore known as "uncooled" IR cameras. They capture the scene using a relative measurement from micro-bolometer arrays. A micro-bolometer is a several microns wide element that changes resistance depending on its temperature (refer to Figure 3). Currently the bolometers are either 46 or 28 microns on a side and are made of vanadium oxide, a substance that is very sensitive to IR radiation. By determining the resistance of each bolometer in the array a representation of the IR energy in the scene can be created.

The Imaging System

The imaging system includes the optics that focuses the IR energy onto the array of micro-bolometers in addition to all the electronics that determine the bolometers'

resistances and process that raw data to make an image. The optics will, for the most part, be ignored as they are the same for both TEC and TEC-Less operation. The electronics are produced on several Circuit Card Assemblies (CCA) that together form the imaging module. The most important CCAs will now be described in more detail.

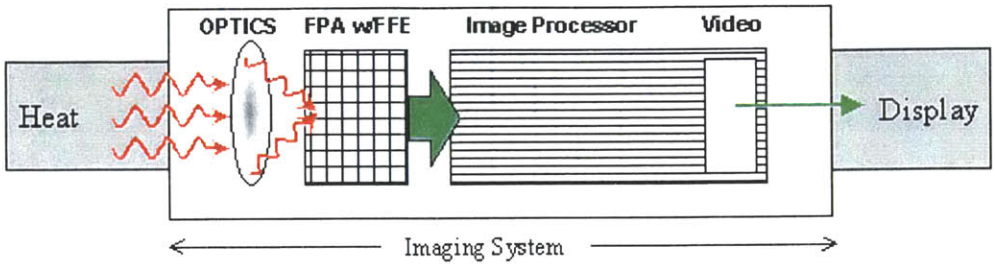


Figure 2: Imaging system block diagram

The Focal Plane Array (FPA) and FFE

The FPA is located in a vacuum sealed package that contains the array of microbolometers on top of a CMOS Read-Out Integrated Circuit (ROIC) and the TEC. The bolomters themselves are one of two sizes, 28 or 46 micons, and the array can also be one of three sizes. The FPA used for the tests was a LAM2D which has 46 micron bolometers, like the one shown below, in a 320x240 array.

Great pains are taken to isolate the bolometers as much as possible from all heat sources including the ROIC and the ambient environment. As can be seen in Figure 3 below each bolometer is suspended above the ROIC by very skinny legs which also serve as the electrical contacts. The legs are as narrow as possible to minimize heat transfer to the ROIC. The package is vacuum sealed and has a Germanium window, very transparent to longwave IR (8-12 microns), directly above the bolometers. This allows the IR radiation coming through the optics to reach the bolometers in the sealed package.

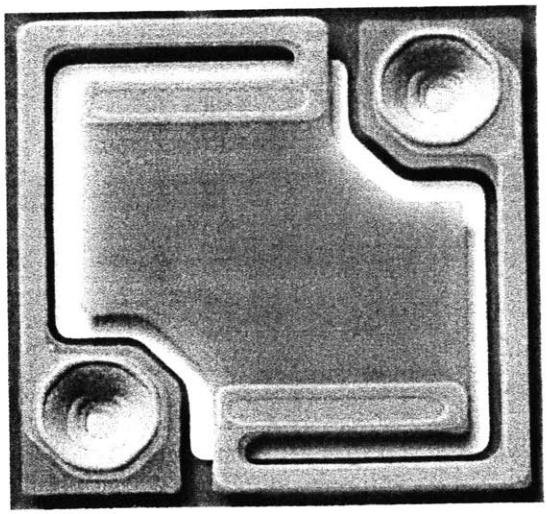


Figure 3: 46um bolometer, note thin contact legs

BOLOMETER	ARRAY SIZE 160x120	320x240	640x480
46 um	SAM	LAM	
28 um	???	CASPER	SPIRIT

Table 1: FPA names with corresponding bolometer and array sizes

The FFE CCA contains the circuitry that supports the ROIC. It provides and filters all the voltages for the ROIC that come from other CCAs, or in some cases are set on the FFE with digital potentiometers such as the bias and offset voltages. It is also responsible for sending data from the ROIC out to the Video Signal Processor (VSP).

The TEC

Even though the bolometers are isolated as much as possible they still sense very small changes in ambient temperature. Therefore the ROIC and array are kept at a constant temperature by the TEC. The TEC can keep the FPA at any given temperature over a large range depending on the TEC Set Point (TECSP). The module is normally

calibrated at a particular TECSP and is then always run at that temperature so that the only change in bolometer resistance is due to radiation coming from the scene. In order to eliminate the need for a TEC, the system must continually update bias and offset voltages as well as correction maps in order to compensate for the unwanted changes in bolometer resistance due to ambient temperature changes.

The ROIC

The ROIC must determine the resistance of all the bolometers, which is 320 x 240 = 76,800 values, and then digitize each value to 14 bits. One time through the array, (76,800 values) constitutes one video frame, which are collected at standard rates: 60 Hz for the NTSC video standard and 50 Hz for the PAL video standard.

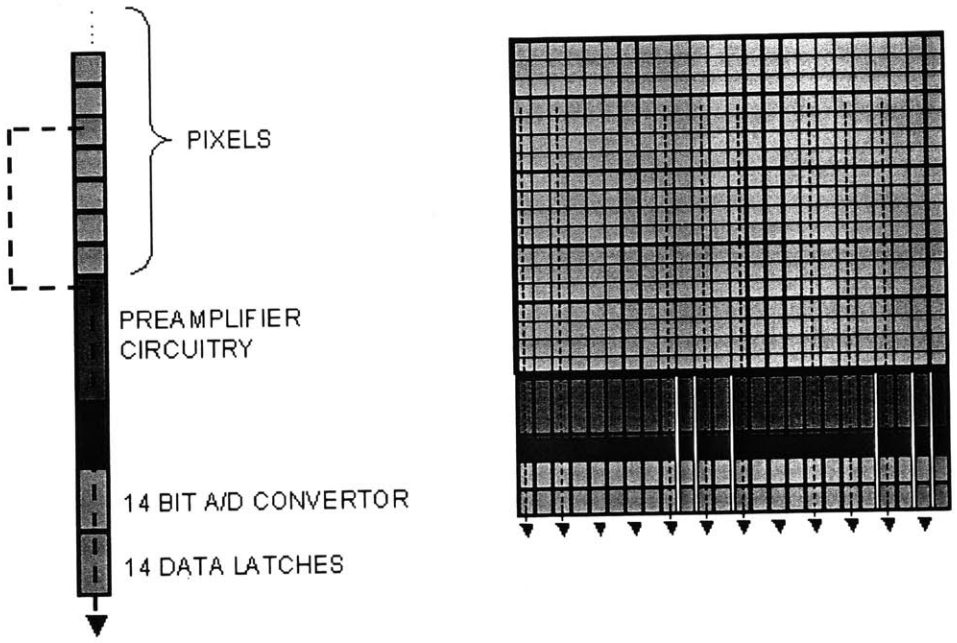


Figure 4: ROIC Block Layout

The ROIC can be considered as a collection of column structures corresponding to each column of bolometers. The ROIC connects one row of the bolometer array to the IC at a time, therefore the bolometers in one column can all use the same pre-amplifier, sample and hold, 14-bit A/D converter, data latches and other column elements. Each element in a row is biased simultaneously with global offset and bias voltages.

Much of how the ROIC operates can be understood by studying the preamplifier and biasing circuits shown below. As can be seen in the circuit, the voltage across and the current going through the bolometer can be controlled directly through several variables. The *Detector Bias* pin sets the voltage across the bolometer. Since *Bias* is a global variable, it is set once and remains the same for all bolometers during normal operation.

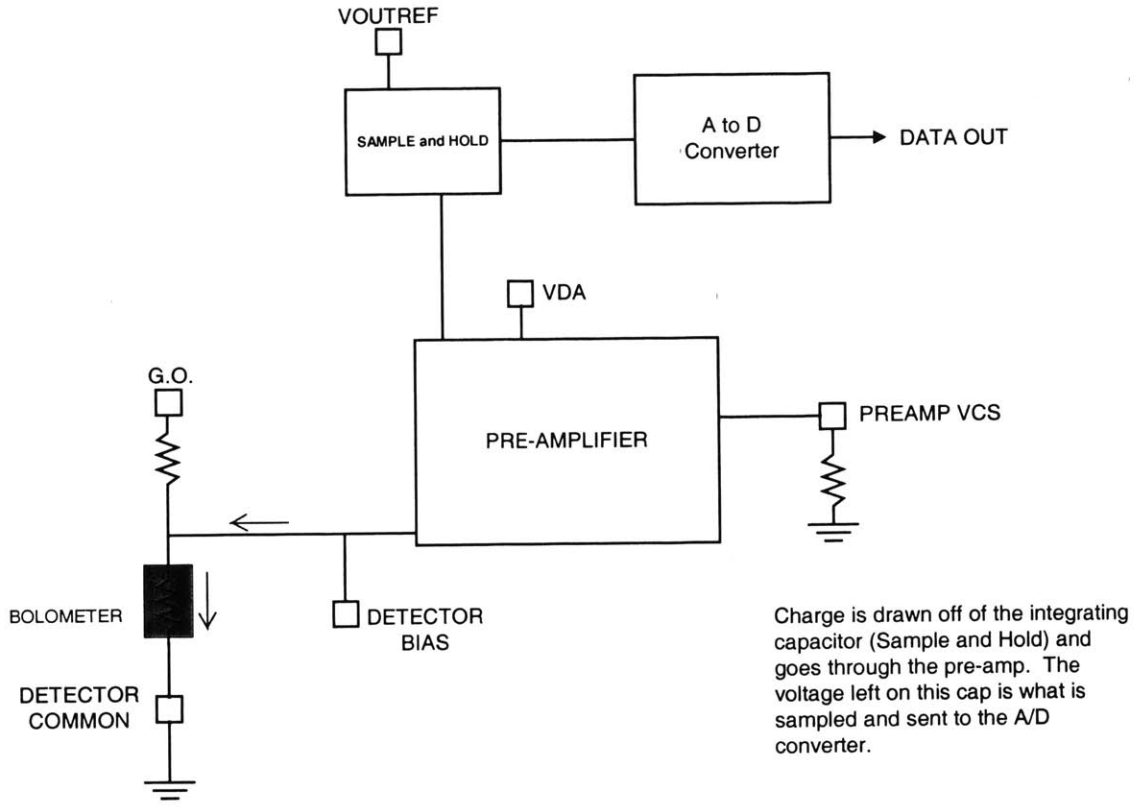


Figure 5: ROIC Circuit

With the bias at a fixed voltage, the current through the detector depends, of course, on its resistance. However, as noted earlier the resistance depends directly on the IR radiation being absorbed. The current being drawn is sensed by the ROIC and is the determined video value for that detector. Most of the current going through the bolometer is sourced through the *Global Offset* (*GO*) pin. The remaining current comes from the integrating capacitor in the sample and hold circuit. The amount of charge drained is directly proportional to the voltage change across the cap. This voltage is then converted to a 14-bit digital value. Using the following equations:

$$Q=CV \text{ and } Q=It$$

Where Q=charge, C=capacitance, I=current and t=time

Combining the equations gives:

$$\Delta V = It/C$$

$$\Delta V = (550nA)\ (55uS)/(12\ pF)$$

$$\Delta V=2.5V$$

Using typical values, we see that the current coming off of the integrating cap caused a voltage drop of 2.5 Volts. Since that side of the cap is nominally set to 9.5V at the beginning of every row time, the final voltage would be 7 volts (9.5V-ΔV). This value is then transferred to the analof to digital converter, which produces a 14-bit value. This value is then sent to the VSP.

Video Signal Processing (VSP CCA)

It is impossible to make identical bolometers with each having the exact same resistance at a given temperature. Each bolometer will react differently to dynamic changes as well. Therefore in order to get good images several corrections have to be applied to the raw data coming from the focal plane. There are two main types of corrections, pixel corrections and global corrections.

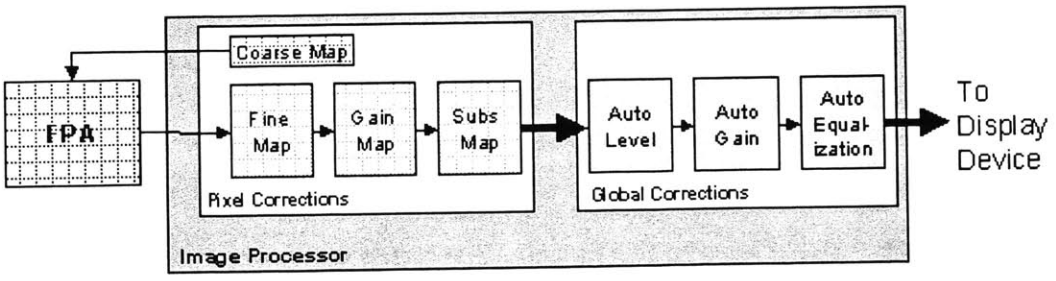


Figure 6: Image Processing corrections

Pixel corrections are done with a series of "maps." There are four maps: coarse, fine, gain and pixel substitution. To correct for the inherent difference in resistances there is an adjustable resistance in series with each bolometer. The processor uses the coarse map to set each of these resistors to the appropriate value. This map is the only one that changes something directly on the FPA. The remaining maps manipulate only the data

coming from the ROIC. The second map is the fine offset map. This map is a stored frame that gets subtracted from every incoming frame and in essence does the same thing as the coarse, adjusts for different resistances, but does the pixel by pixel adjustment by data subtraction, not by changing physical resistances.

The third map, Gain, adjusts for differences in response. That is, a pixel may change more drastically given a change in temperature than the one next to it. The processor applies a gain multiplier to each pixel in every frame which compensates for different dynamic changes in the bolometers. The final pixel correction map is the substitution map. If a bolometer has no or undesirable response to IR inputs, the processor will replace that pixel's digital value with the value of one of its neighbors in the video output.

Global corrections, also known as Automatic Error Correction (AEC), are applied to an entire frame of data, not to individual pixels. There are three global corrections. The first, Auto level, adjusts the video so that the average output value is at mid-scale. The second, Auto Gain, tries to use the largest range of values possible for a scene by setting the coldest object to black and the warmest to white. The last, Auto Equalization, uses non-linear processing and histogram equalization to allow viewing of the most pixels. For example, a very hot small object in a scene would hide all other details if histogram equalization were not used.

Test Procedures

The tests were conducted using the setup shown in Figure 7. A majority of the testing was conducted outside of the thermal chamber in order to save time, but final tests were conducted in the chamber. The TEC was used to adjust the FPA temperature just as if the ambient temperature was changing when outside the chamber. This was the preferred method because the TEC changes the FPA temperature much more quickly than ambient temperatures in a thermal chamber. Since the TEC was used to control FPA temperature, some graphs in the test results may show TECSP (cnts), *TempOut* (Voltage), or degrees Celsius as units of temperature measurements. When possible, all data was converted to actual Degrees C.

The exact FPA temperature was determined by monitoring the *TempOut* pin of the FPA. The voltage on this pin has a temperature coefficient of –21.4 mV/degC. This correlation was determined empirically by monitoring the voltage while the unit soaked at different temperatures in the thermal chamber. The results are graphed below in Figure 8 with the linear fit equation shown on the graph. This is the equation used to convert *TempOut* voltage to degrees Celsius for all tests where *TempOut* voltages where collected.

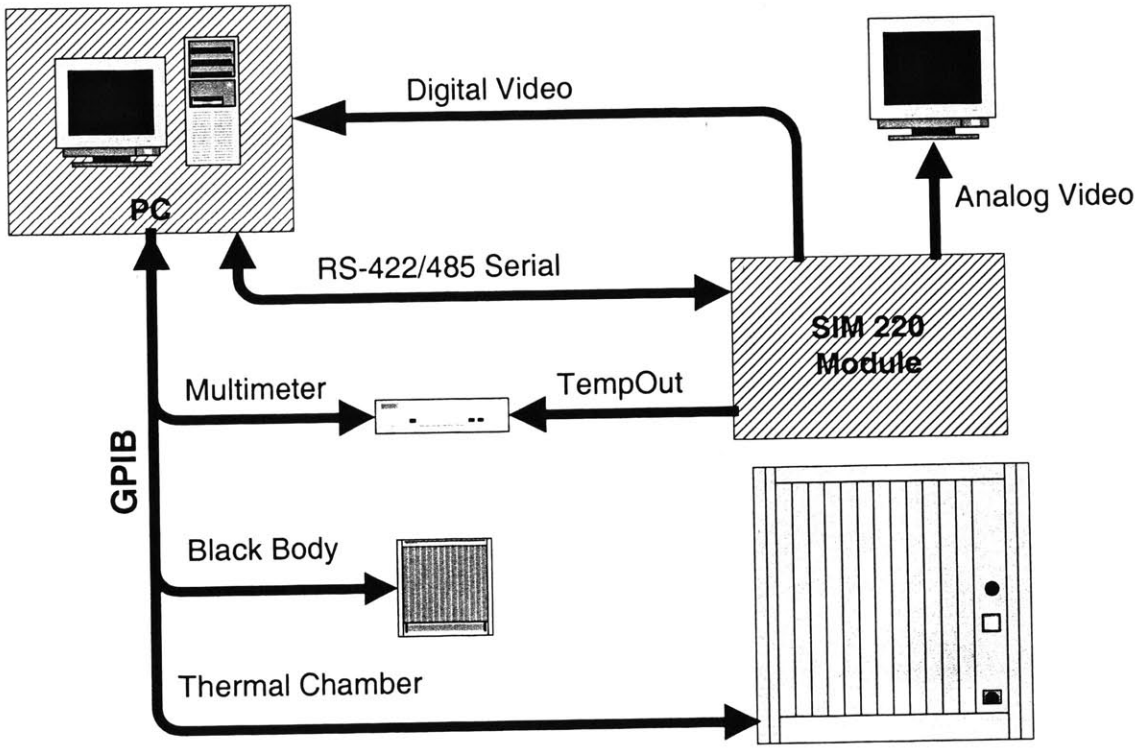


Figure 7: Test bench setup block diagram

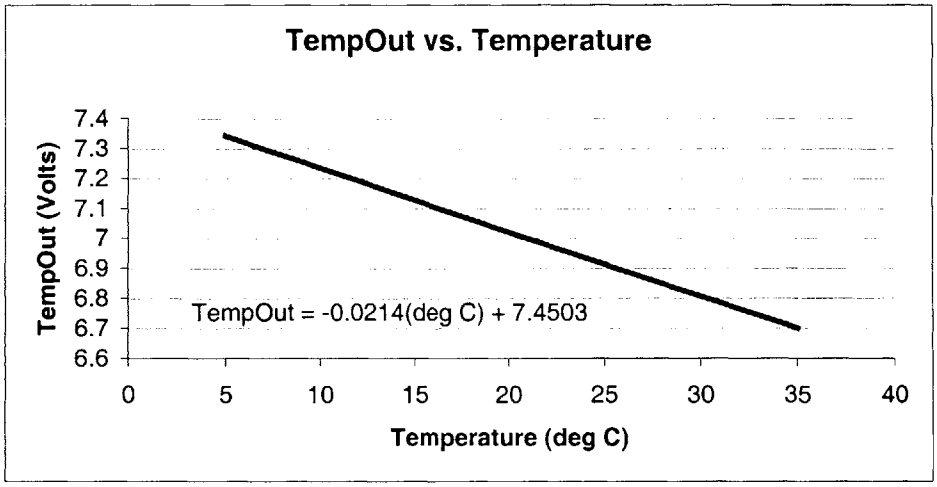


Figure 8: *TempOut* Vs. Temperature

All system commands were sent over a serial connection from the PC. Any software changes were implemented in PC software in C++. Any required map updates were downloaded to the module. This gave a lot more freedom of implementation than changing the embedded code on the unit. However, many of the tests required changing the fine correction map, which was difficult using the PC due to long download times. The average time for download was 35 seconds so tests that tried to update the maps often took a long time and results could have been skewed because of this delay.

Unless specified otherwise it can be assumed that the coarse and gain maps are constant during any given test. The focus of most tests was either changing the *Global Offset* and *Bias* voltages in some way or updating the fine correction map. Specifics on gain and coarse will be given later.

All tests were done while the unit was focused on a uniform field. When the unit was outside the chamber, the field was provided by a uniform Black Body (BB). The BB will create a uniform field at a given temperature, which for most tests was 55°C. There was no room to use the BB while the unit was in the chamber, so the uniform field was created by a mat of black rubber at the same temperature as the chamber set point. A uniform field was critical as the purpose of the tests was to determine how the bolometers change due to ambient temperature changes, therefore all changes due to the scene temperature had to be controlled and eliminated.

Both the BB and thermal chamber were controlled by the GPIB, allowing automated tests to be conducted. The digital video was collected by a frame grabber card in the computer.

TEC-Less Operation

The challenge of running the module without a TEC is keeping all the bolometers within range (average value) as well as keeping them from drifting away from each other (deviation). The video values collected by the frame grabber are 16-bit digital values. So each pixel can have a value from 0 to 65536 (2^16). While the camera is focused on a uniform temperature scene the ideal output is to have all pixels at mid-scale, a value of 32768, because this allows for the most dynamic range. However, the amount of deviation is most important since higher deviation makes the image look worse.

As an example, the system coarse and gain maps were calibrated at a temperature of 16.5 °C, the temperature changed, and the video counts examined. The video histogram values are shown in Figure 9. Notice that the average video value changes with temperature, which is the expected response of the bolometers. More importantly, the histogram spreads out at temperatures away from calibration causing a degraded image. The standard deviation is many times greater with temperature changes less than a degree. A second point of interest is that the histogram spreads out much more for a given increase in temperature compared to the same decrease.

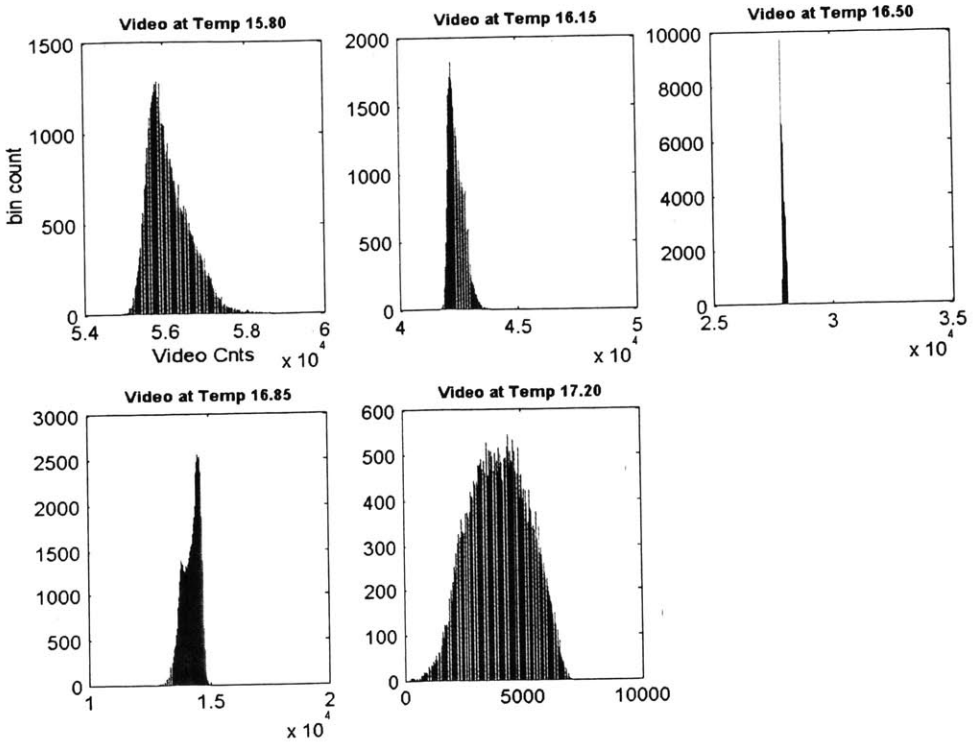


Figure 9: Video Histograms at several temperatures. Note widening of histogram over temp.

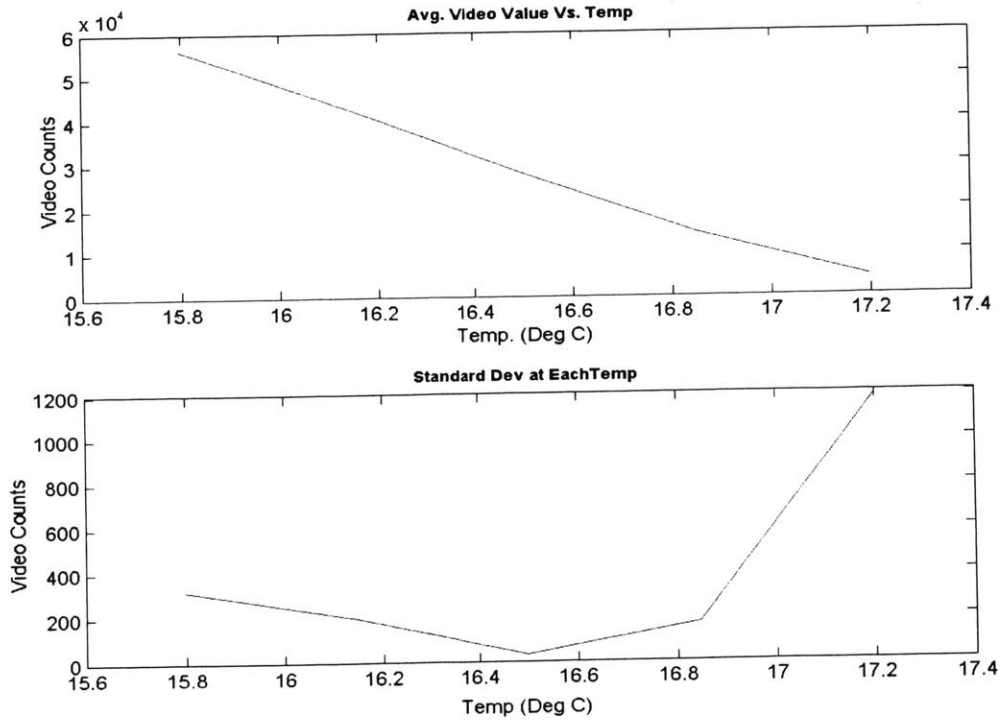


Figure 10: Average and standard deviation of histograms shown in Fig. 9

Global Offset Adjustment

In order to bring the average video value back to mid-scale, the *Global Offset* voltage (*GO*) must be adjusted. As mentioned before *GO* sources most of the current drawn by the bolometers. Therefore by adjusting the voltage on this pin the amount of current can be set so that the rest of the current drawn from the integrating capacitor yeilds mid-scale values (refer to Figure 5). *GO* is usually set by a digital pot to a single value and does not change during normal operation. Therefore a new circuit was needed to continually change *GO* during TEC-Less operation. Both an analog feedback circuit and a digital approach were explored. The digital circuit worked better and had the advantage that it could be paused, holding *GO* constant at any time.

Analog GO Adjustment

An analog circuit was used to monitor *Voutref*, the voltage connected to the positive side of the integrating capacitor, and adjust *GO* accordingly. By sensing small changes (a few mV) in this voltage it can be determined if too much or too little current is being drawn from the integrating capacitor. The feedback circuit continually adjusts *GO* such that *Voutref* is at the correct level. However, there are several problems with the analog approach. First is that the sensed change in *Voutref* is very small and the circuit is therefore very susceptible to noise. Secondly, the analog circuit cannot be turned off easily. It is continuously changing *GO*, which causes problems when calibrating the unit and performing tests.

The noise problem with the analog feedback approach was the limiting factor. To demonstrate this point the analog circuit was connected while the TEC was on and held at a constant temperature, so *GO* should theoretically be constant. The resulting *GO* voltage and video frame average are graphed in Figure 11. While there are only slight changes in *GO* voltage, the changes in frame average are hundreds of counts, which is very noticeable in the video output.

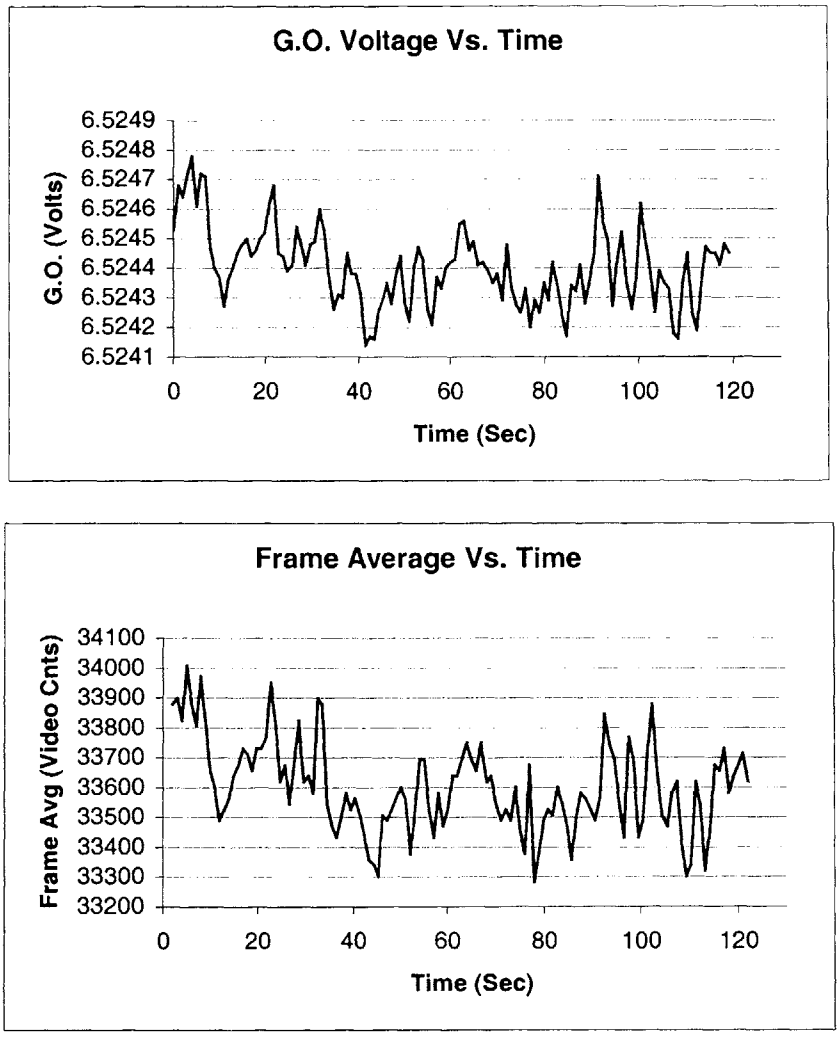


Figure 11: *GO* Voltage Vs. time and Frame Avg. Vs time. Note large change in Frame average that follows small change in *GO*

Even with the inherent noise the *GO* adjustment does succeed in keeping the video average near a particular target value and by doing so greatly improves the deviation of the pixels across temperature. In Figures 12 and 13 the video histogram and standard deviations of frames taken with the analog circuit running are shown. Notice the large decrease in deviation from what was shown earlier in Figure 10. Again if not for the noise causing changes in the frame average, this would be a preferred approach.

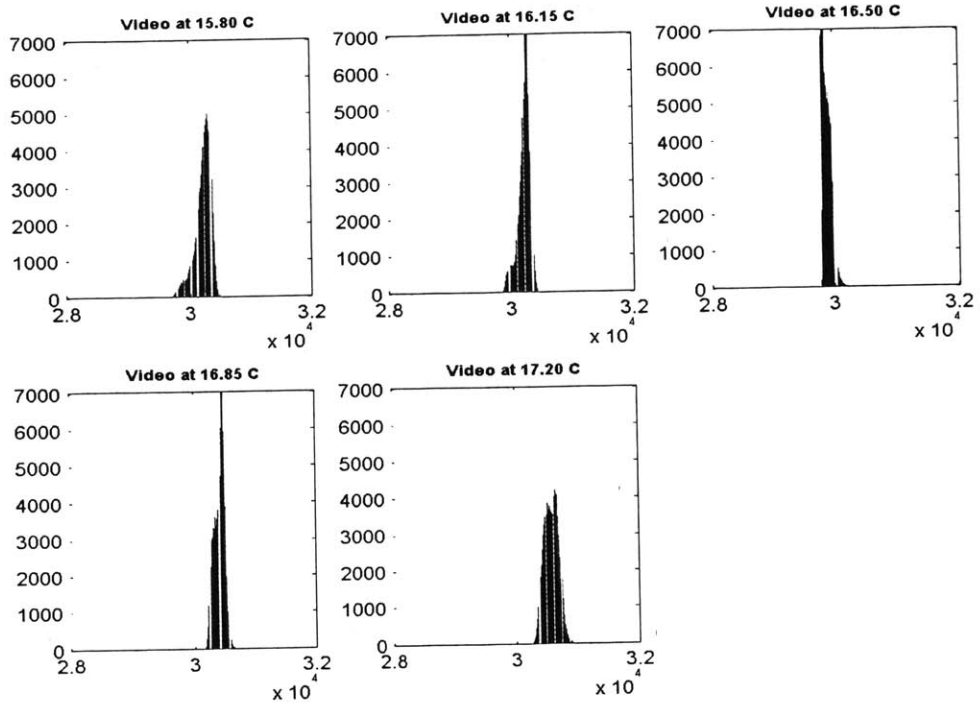


Figure 12: Video histograms over temperature with analog *GO* adjustment on

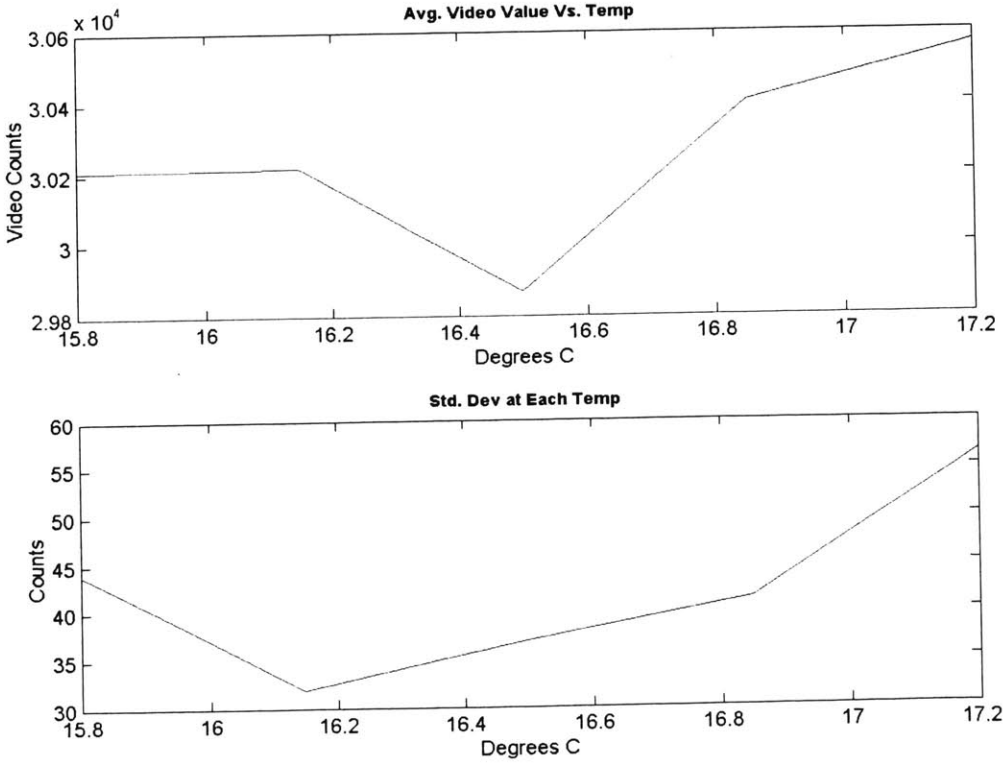


Figure 13: Average and standard deviation of histograms in Fig 12

Digital GO Adjustment

To overcome the large jumps in frame average caused by noise in the analog circuit, the digital circuit shown in Figure 14 was implemented. By monitoring the frame average and adjusting the three digital pots accordingly, the frame average could be held at a near constant value. The algorithm that monitors the frame average and adjusts the pots is shown in Figure 15.

In order to implement this algorithm the step change in video counts for a given change in the Coarse, Fine and Xfine pots is needed. However, these step values are functions of FPA temperature. The unit was run at several temperatures and at each the average change in video counts per step in each pot was determined. Those averages are plotted in Figure 16 for the Fine and Xfine pots and a slope determined. The counts per step are 32,000 counts for Coarse, 200 for the Fine and –1 for the XFine. It should be noted that while testing at very cold (< 0 degrees) and very warm temperatures (> 55 degrees) the linear term of the equations for the step change had to be modified in order for the algorithm to work correctly. More testing over the entire temperature range would most likely lead to suitable equations that could be used over all temperatures.

The algorithm works off the actual scene average and a predicted scene average. Referring to Figure 15, the algorithm computes the difference between the actual scene average and the target value. It then computes the number of steps of the Xfine pot needed, using the slope determined above, to correct for the difference. If the number of steps causes the Xfine pot to go out of range it sets the Xfine pot to mid-scale. The algorithm next computes a predicted scene average, which takes into account the change due to setting the Xfine to mid-scale. It then uses this value in calculating the number of Fine pot steps needed to adjust for the difference between the predicted scene average and the target value. A similar routine is then done for the course pot. The routine loops through these instructions until the Xfine correction is small enough to remain in range,at which point it ends.

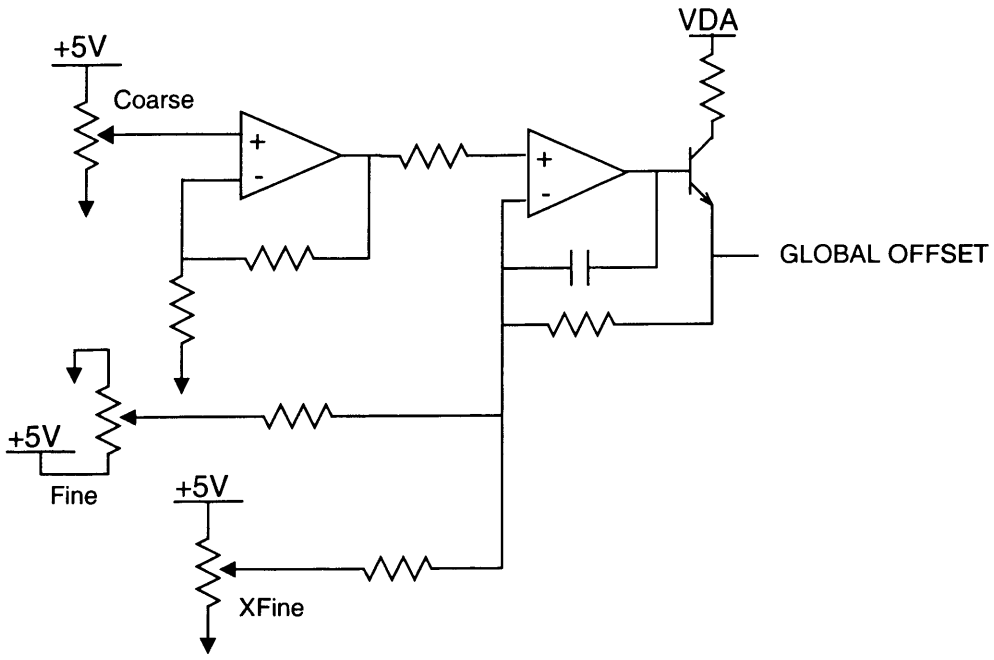


Figure 14: Digital *GO* adjustment circuit

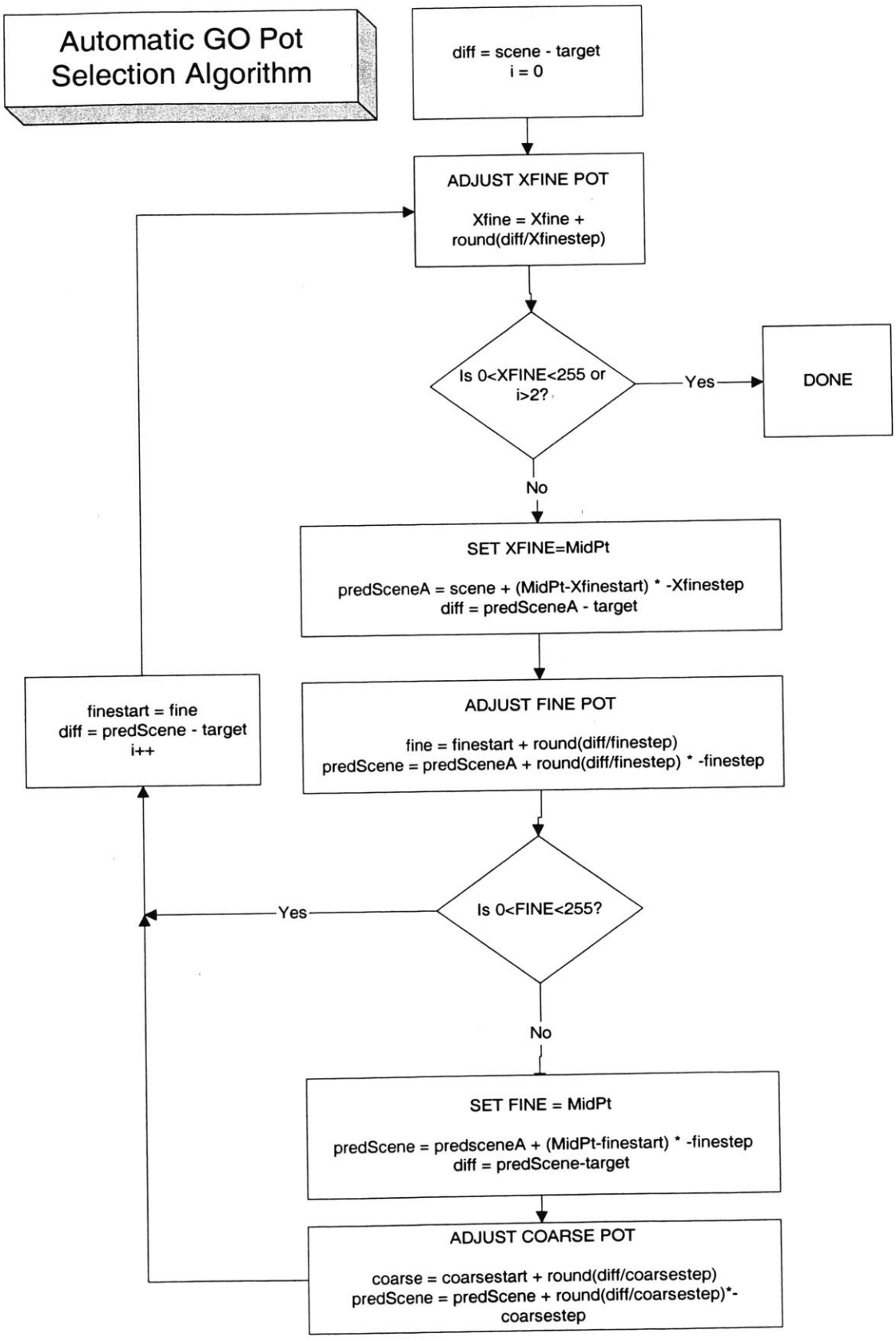
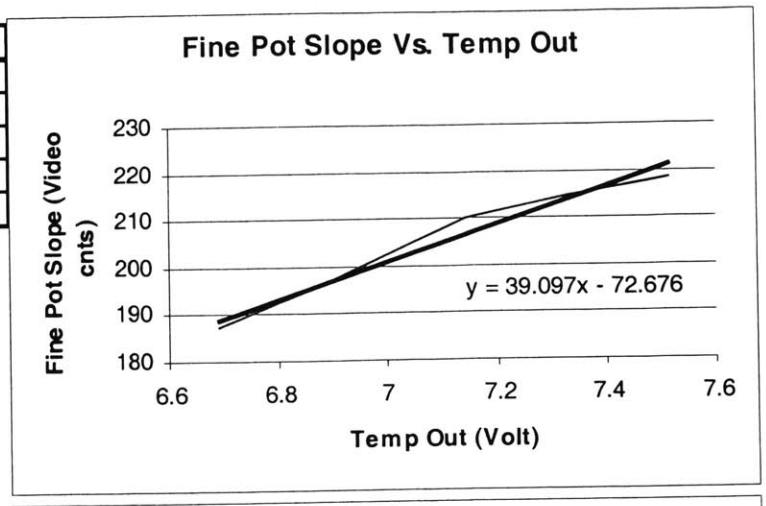


Figure 15: Digital *GO* adjustment algorithm flow chart

Temp Out	Fine Slope
7.51376	218.4695229
7.32628	214.8684816
7.14046	209.861634
6.88041	196.0044132
6.69247	187.4436438



Temp Out	Xfine Slope
7.51374	-1.718141176
7.32622	-1.052465158
7.10316	-0.99919819
6.88035	-1.134600905
6.69239	-1.085469683

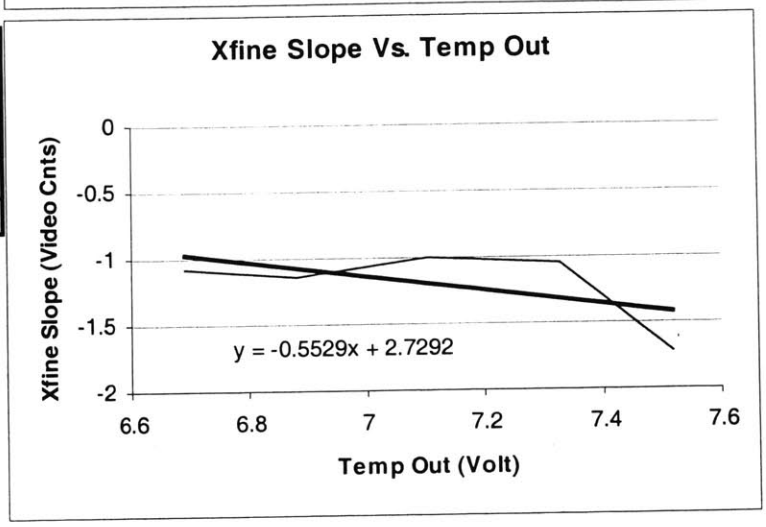


Figure 16: Fine and XFine pot step changes Vs. TempOut with associated tables

The digital circuit performs much better at maintaining a constant scene average at a given temperature than the analog circuit. Figure 17 below shows the frame average plotted against the number of iterations through the *GO* adjustment algorithm. The first graph shows the frame average jumping down at each step in temperature but then quickly coming back to the target value after a few times through the algorithm. The graph in Figure 18 shows this change more closely. It can be seen that the scene average returns to within 15 counts of the target value after three iterations from the .4 degree change, and remains within 15 counts of the target value.

The algorithm was run on the PC while collecting frames from and sending commands to the unit. Because of the delay from these communications the procedure as

tested could only update *GO* approximately every 4 seconds. The delay is due in large part to the way *Tempout* was measured. This voltage was measured with a multimeter and transferred to the computer through the GPIB port. GPIB is a very slow method for obtaining this value. If this algorithm were implemented in the embedded system the updates could theoretically be much faster, approximately every 6 frames of data or every tenth of a second. If the pots could be updated this quickly the small discrete changes in scene average should be mostly unnoticeable to the viewer of the image.

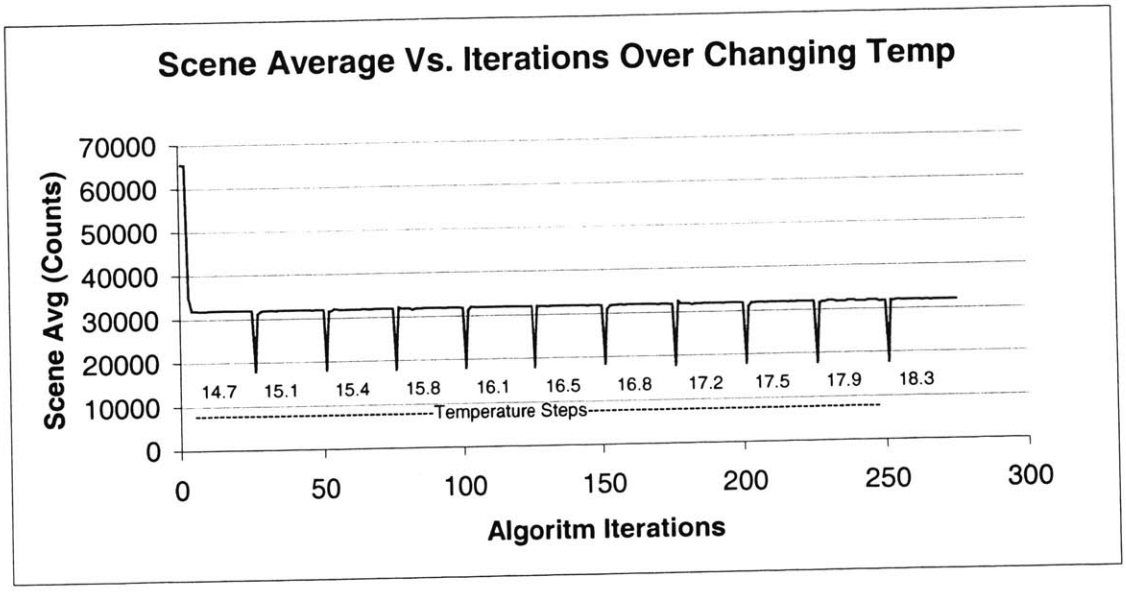


Figure 17: Scene Avg. plotted against number of iterations through digital algorithm. Temperature changes shown on graph

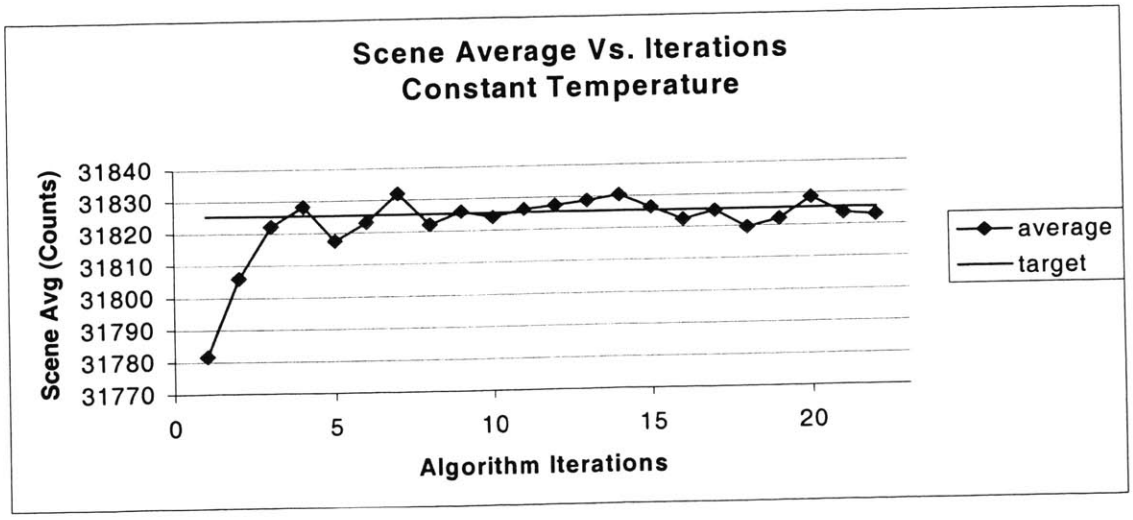


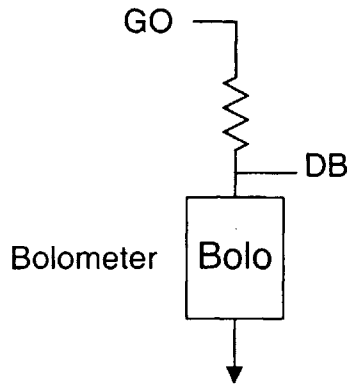
Figure 18: Detailed view of one temperature setting in fig.17

In summary the digital approach was the better of the two methods. Theoretically the analog circuit is nice because it does not take any processing resources and continually adjusts *GO*. However, the change in *Voutref* is too small causing the circuit to be too susceptible to noise or alternatively very expensive. The digital circuit on the other hand is beneficial because it is more accurate and can be turned off for testing and calibration. As long as the pots are updated with enough frequency the discrete adjustments should not be noticeable in the image.

Constant Power

The power dissipated in each bolometer, P = V*I, is a function of *the Detector Bias* and the current being supplied by *GO*. As *GO* is varied to keep a constant frame average the power dissipated varies greatly. Keeping the power dissipation constant is desirable since it helps make changes in pixel deviations over temperature more predictable. That is, if an increase in temperature could be made to produce the same amount of deviation as a decrease in temperature, prediction becomes much easier. As was noted earlier, this is not the case without any power adjustment.

Constant power can be obtained by changing the detector bias inversely to changes in *GO*, and keeping their product constant.



Assume the bolometer has a typical value of 20k and that bias is set at 1 Volt:

Bolometer current: I_B = 50 uA

Power: P = 50 uW

Now assume that the bolometer changes resistance to 18k:

Bolometer current: I_B = 1V/18k = 55.55 uA

Power: P = 55.55 uW

ΔGO to supply extra current:

5.55 uA * 50k = .277 V

ΔDB to return to 50 uW of Power:

50 uW = .90V * 55.55uW or ΔDB = -.1V

Therefore:

ΔDB / ΔGO = -.1 / .277 = -18k/50k

Figure 19: Constant Power example

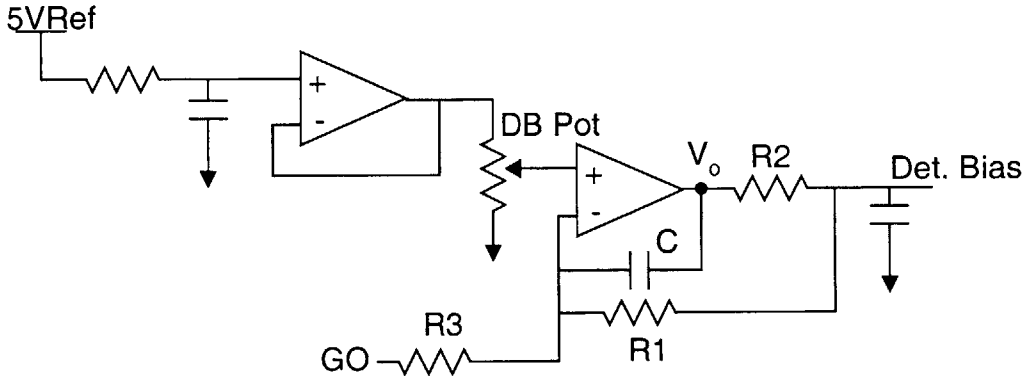
As demonstrated in the above example, the change in detector bias needs be the same fraction as the bolometer to DAC resistance times the change in *GO*, or:

$$\Delta DB = (\text{Bolometer resistance} / \text{DAC resistance}) \times \Delta GO$$

or

$$\Delta DB / \Delta GO = \text{Bolometer/DAC}$$

If this is always the case then constant power will be maintained. The way that *DB* is controlled can be seen in Figure 20. Here *GO* is tied to the *DB* circuit by a single resistor, R3. The value of this resistor sets the value of $\Delta DB / \Delta GO$ to a single value. However, the ratio of DAC resistance to bolometer resistance will of course not be constant. Therefore power dissipation will not be constant but linear with temperature change with this implementation. This is still a great improvement over no adjustment at all.



$$\frac{DB - V_-}{R_1} + \frac{V_o - V_-}{1/Cs} + \frac{GO - V_-}{R_3} = 0$$

$$\frac{CR_1}{R_1+R_3}\frac{dV_-}{dt} - \frac{CR_1}{1/Cs}\frac{dV_o}{dt} - \frac{R_1\ GO}{R_3} = DB$$

$$\triangle DB = -\frac{R_1}{R_3}\triangle GO$$

$$\frac{\triangle DB}{\triangle GO} = -\frac{R_1}{R_3}$$

Figure 20: Constant Power Circuit and derivation

The correct value of R3 was determined experimentally, with values chosen to give a R1/R3 ratio of .1, .3 and .4. The main purpose of R3 is to make the deviation of the bolometers across temperature more even. The following graphs show that this is indeed the case. Figure 21 shows the results with R1/R3 = .1 and no *GO* adjustment is used. Again histograms are shown over temperature as before. For each ratio there are two sets of graphs, one without *GO* adjustment and one with *GO* adjustment. The important thing to note is the symmetry of the deviation between an increase and a decrease in temperature.

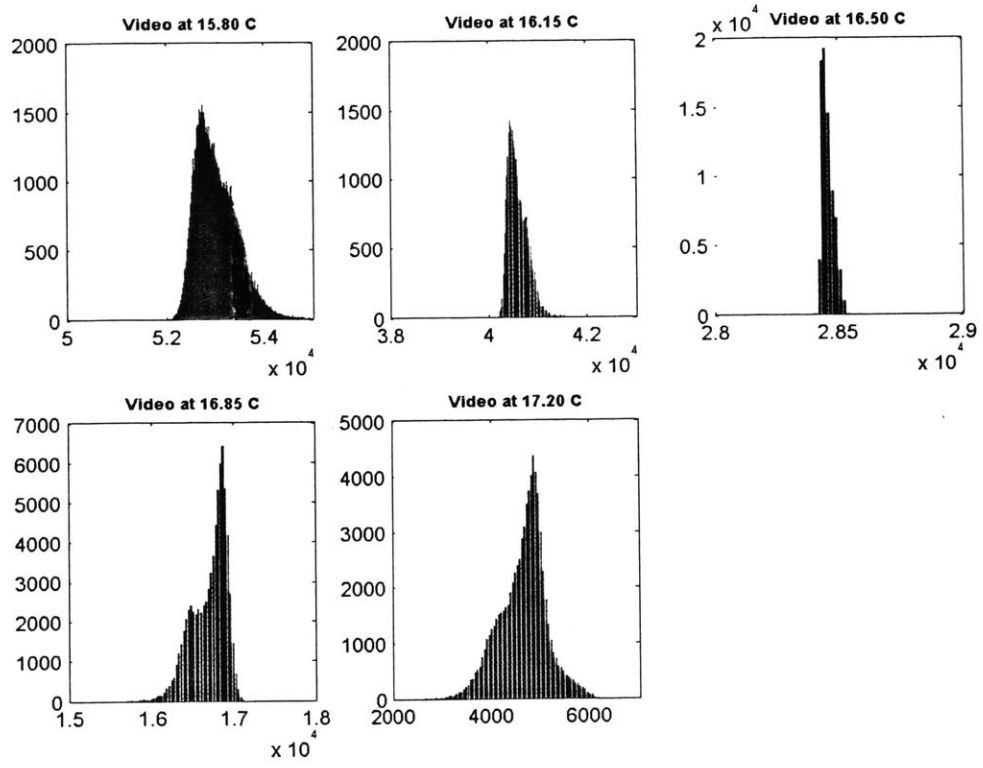


Figure 21: Video histograms across temperature with R1/R3 = .1

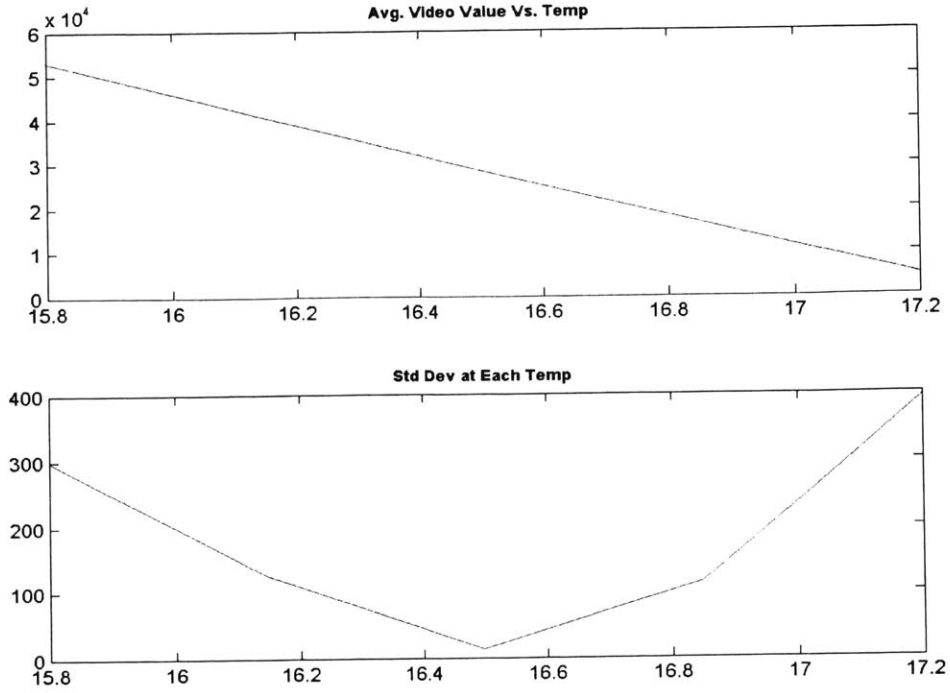


Figure 22: Video Average and standard deviation of histograms in fig. 21

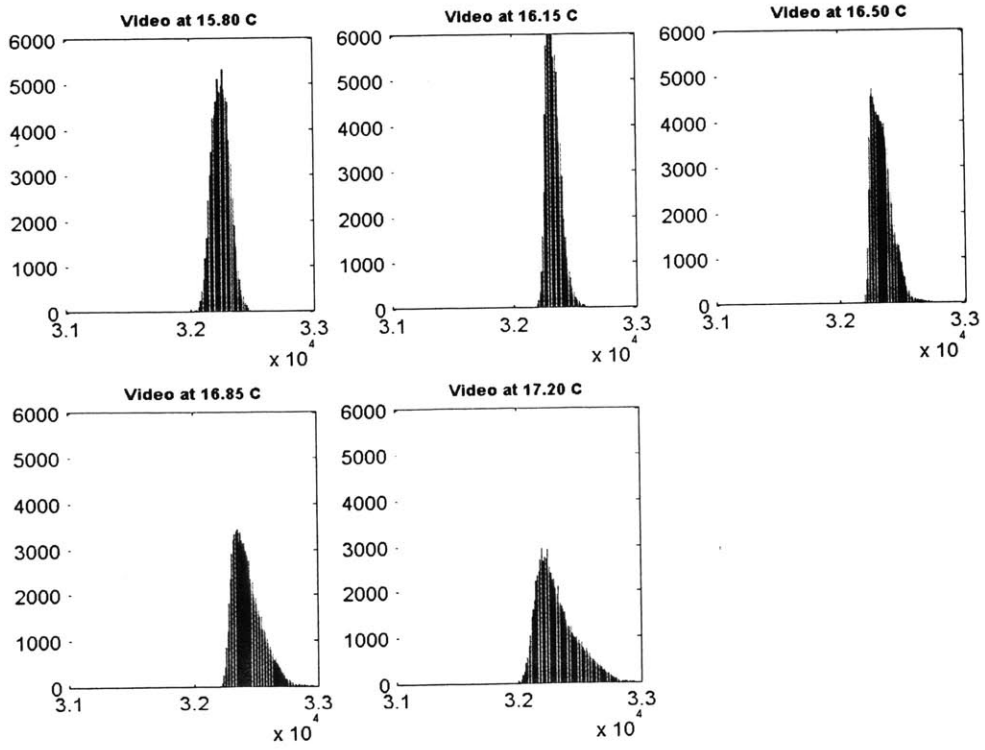


Figure 23: Video histograms over temperature with R1/R3 = .1 and *GO* adjust on

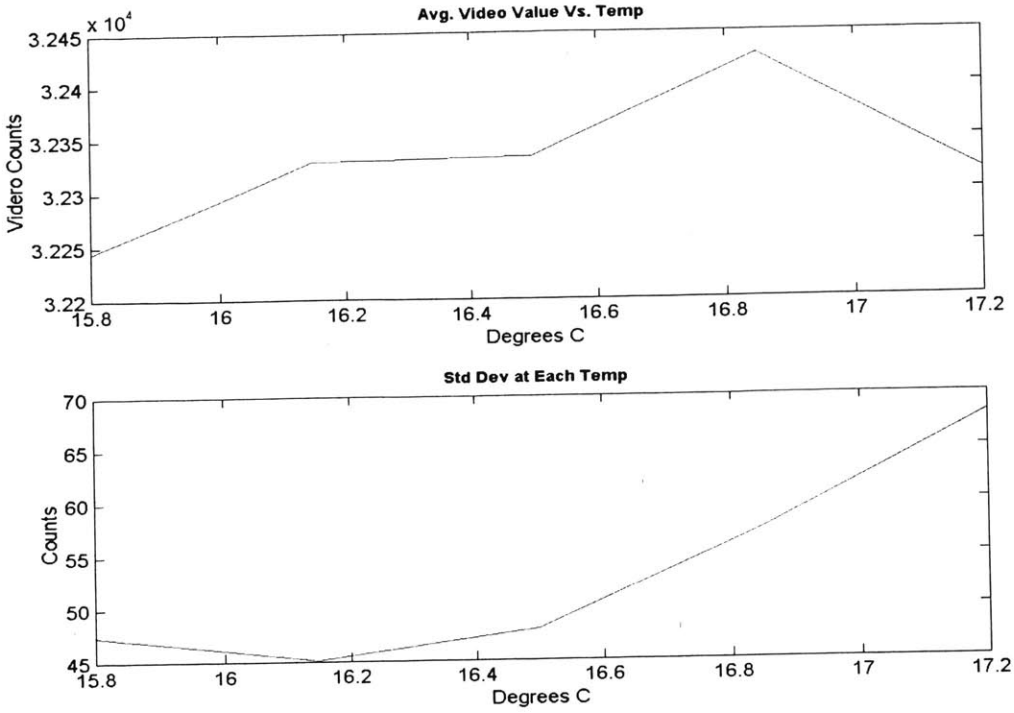


Figure 24: Video average and standard deviation of histograms in fig. 23

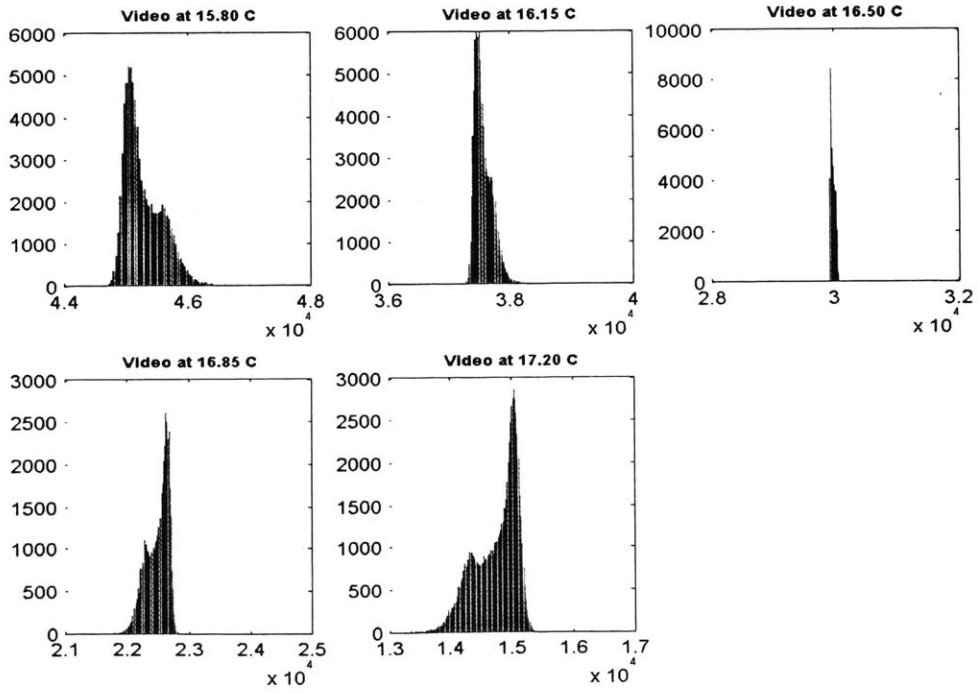


Figure 25: Video histograms over temperature with R1/R3 = .3 and no *GO* adjustment

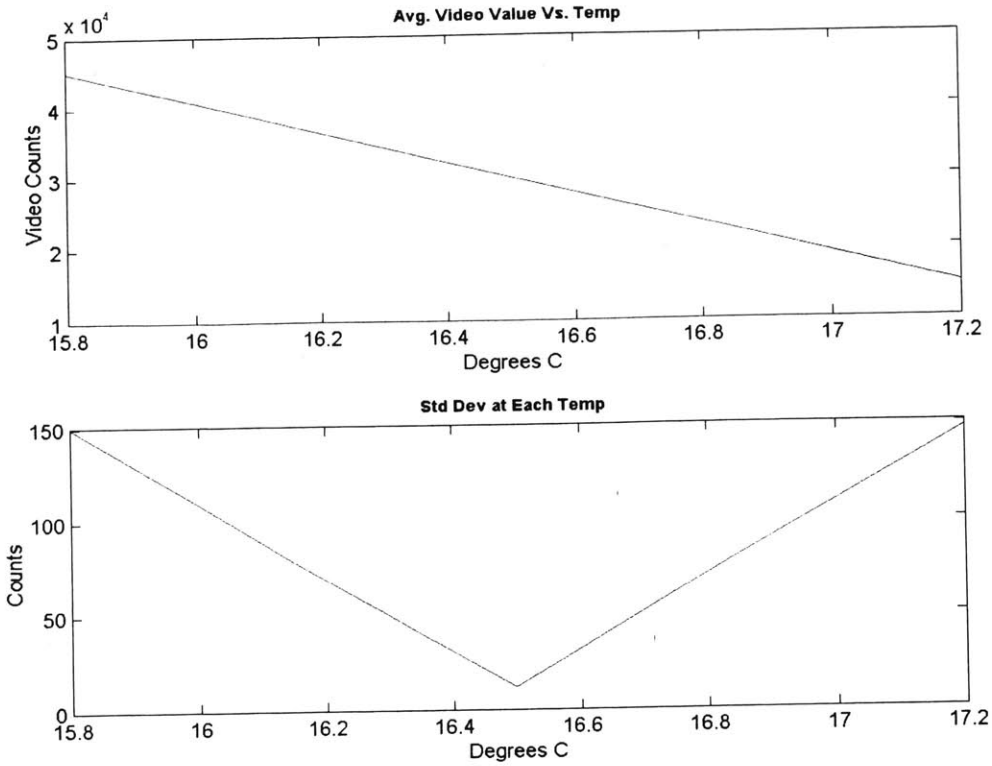


Figure 26: Video average and standard deviation of histograms in fig. 25

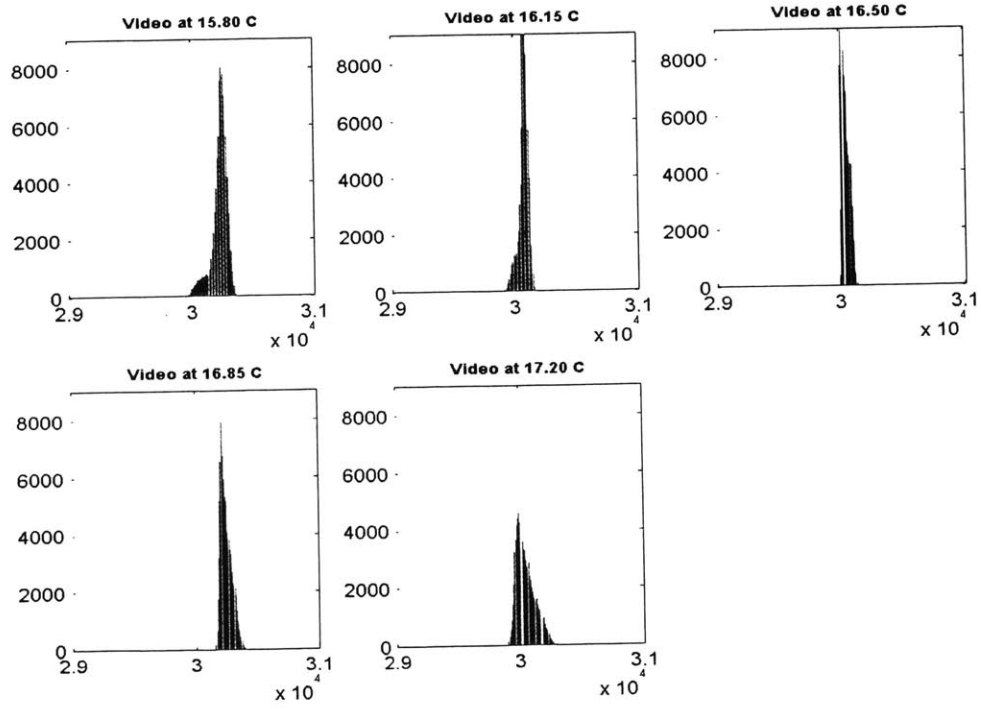


Figure 27: Video histograms over temperature with R1/R3 = .3 and *GO* adjustment on

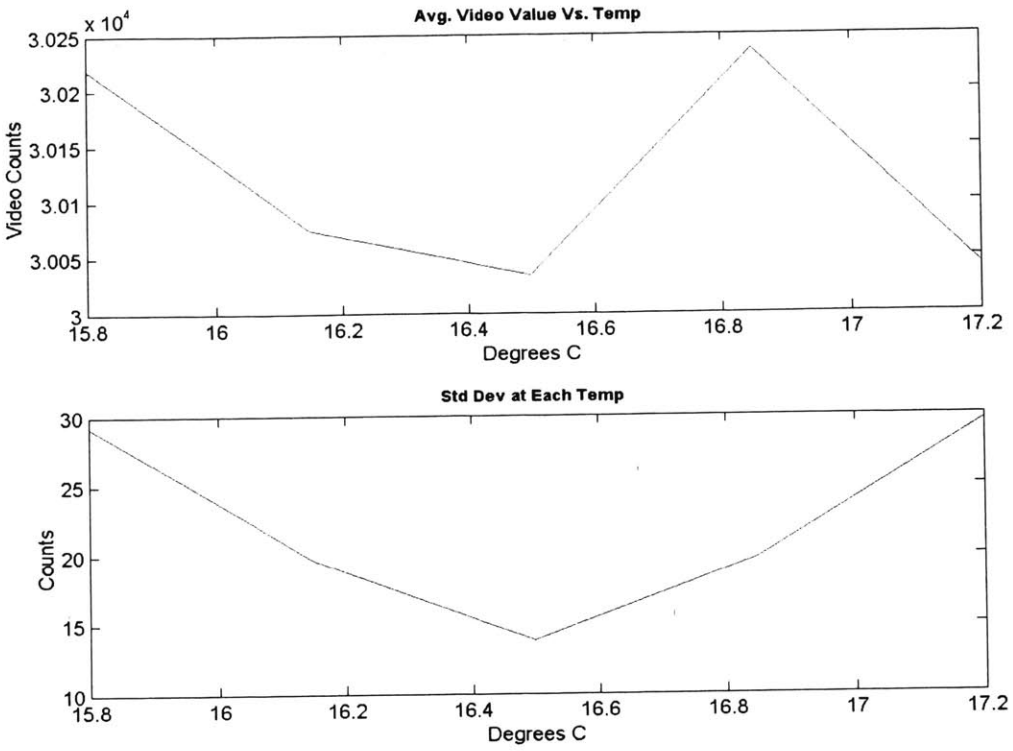


Figure 28: Video Average and standard deviation of histograms on fig. 27

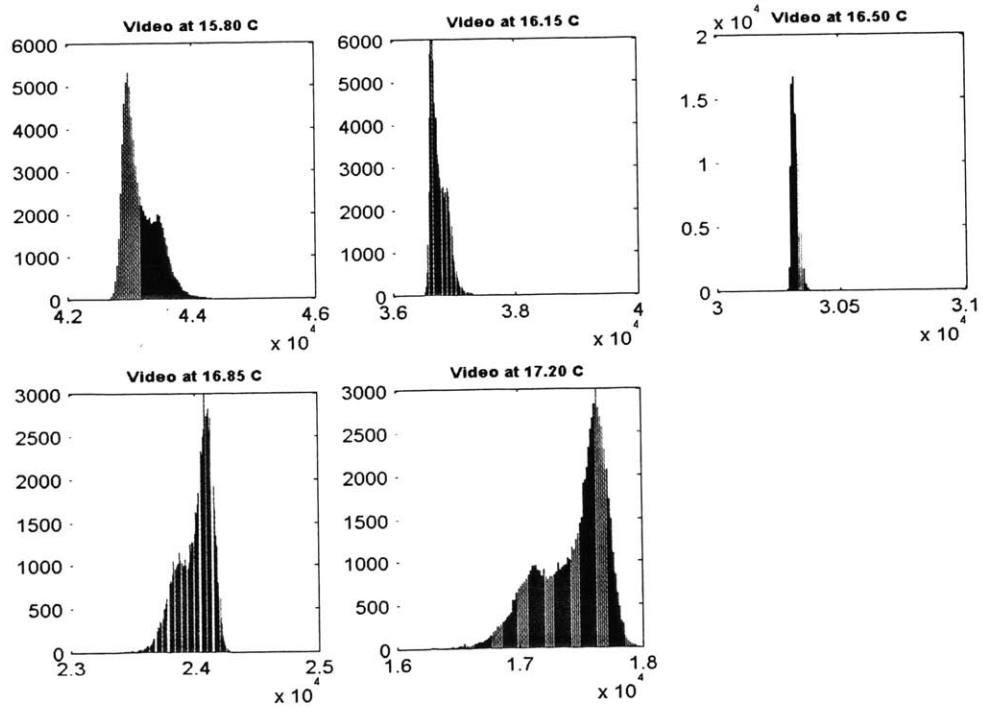


Figure 29: Video histograms over temperature with R1/R3 = .4 and no *GO* Adjustment

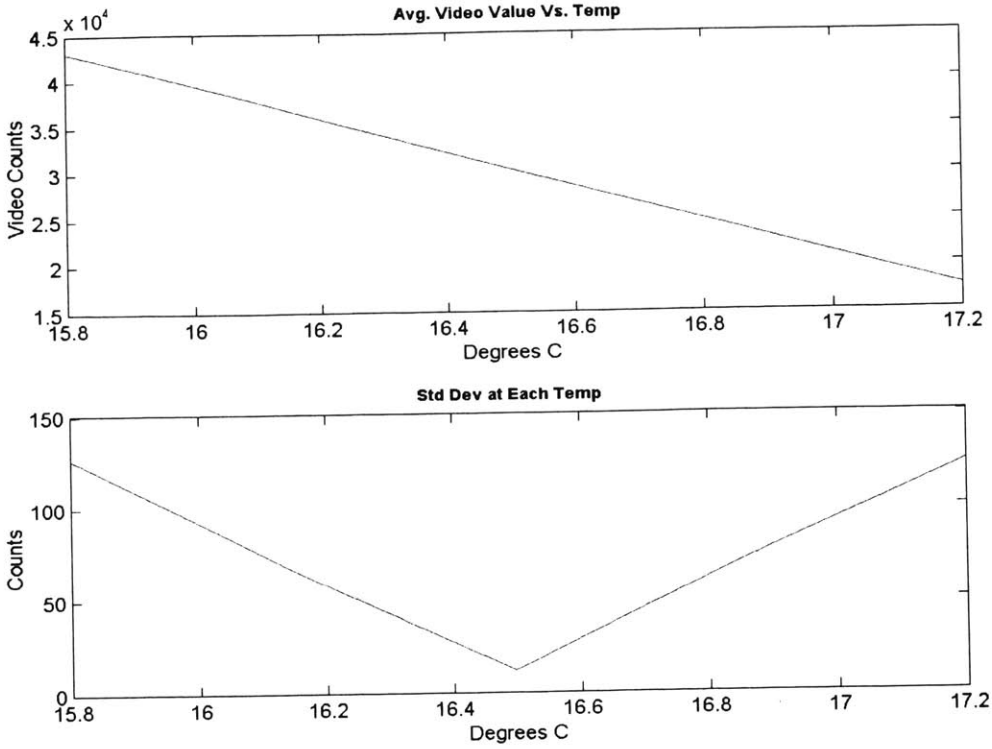


Figure 30: Video average and standard deviation of histograms in fig. 29

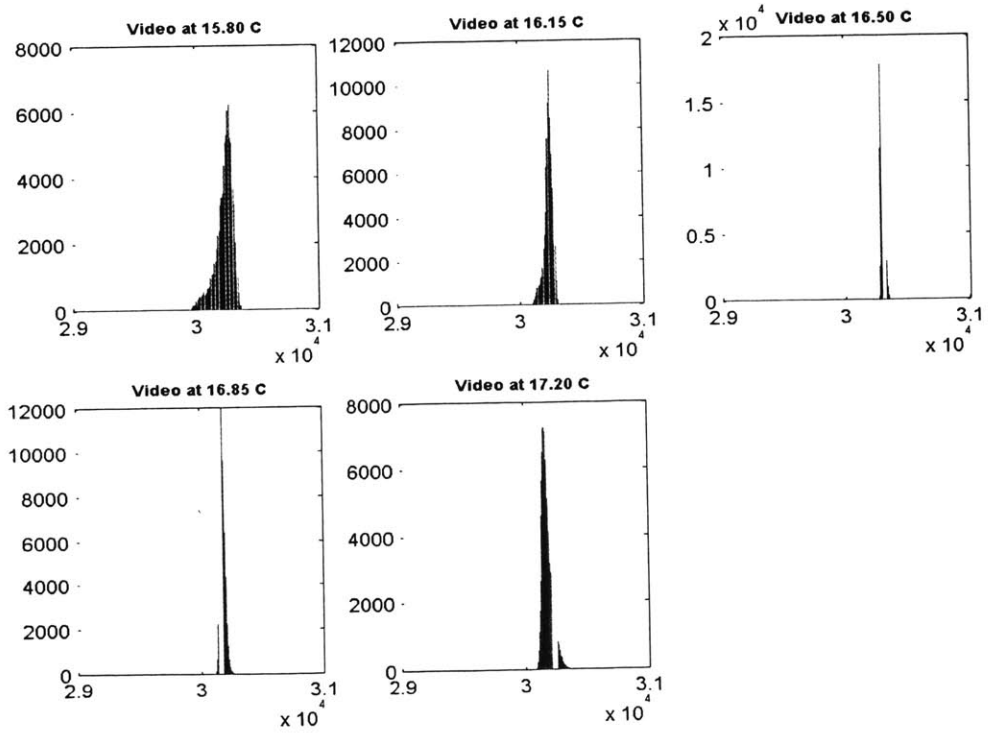


Figure 31: Video histograms over temperature with R1/R3 = .4 and *GO* adjustment on

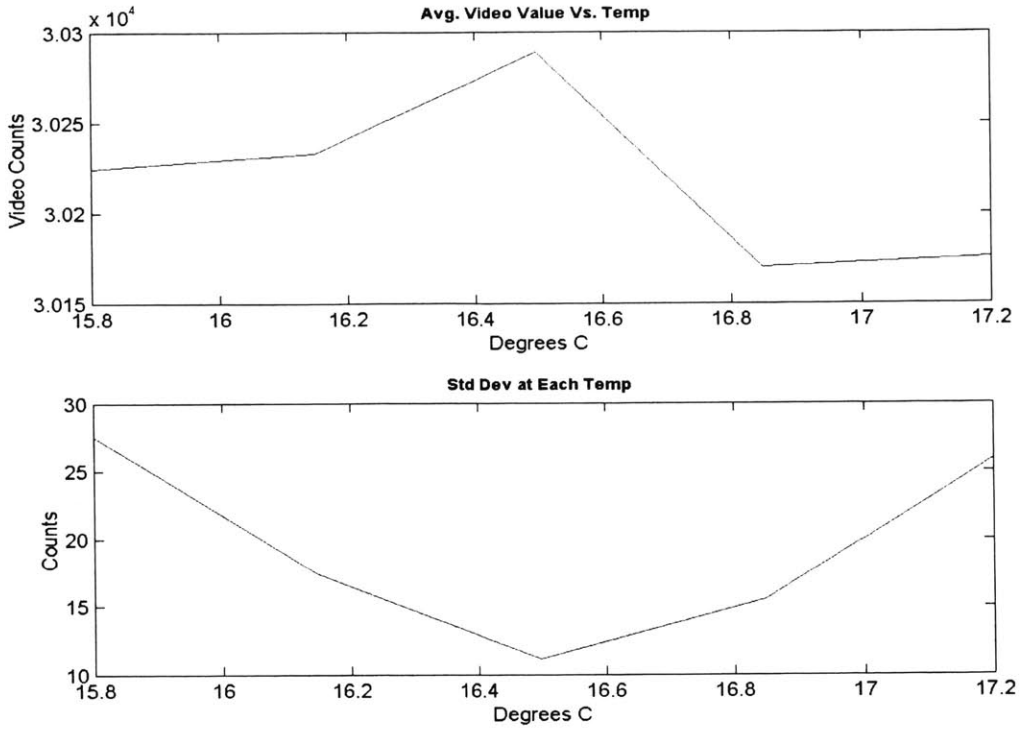


Figure 32: Video average and standard deviation of histograms shown in fig. 31

As can be seen by looking at the standard deviation graphs, R1/R3 = .4 is the best choice of those tested. Figure 33 below supports this assumption further. It contains plots of the standard deviation of fine maps taken over temperature. More detail will be given later as to the purpose and implementation of the fine map, but it is essentially an average frame taken while looking at a uniform scene. This frame is then subtracted from all subsequent video frames in order to compensate for slight variations between pixels.

Each line on the plot is labeled with a number representing the TECSP during calibration. Notice that the fine map has the least deviation when the temperature is near this calibration point (Coarse TECSP). For example, when the system was calibrated at TECSP = 200 the minimal deviation occurs at TECSP = 197 which can be seen on the graph. What is most important is that going either direction from a minimum on any given line results in the same rate of change. That is, going up in temperature increases the variation of the fine map the same amount as going down in temperature. This is due

to the constant power connection.

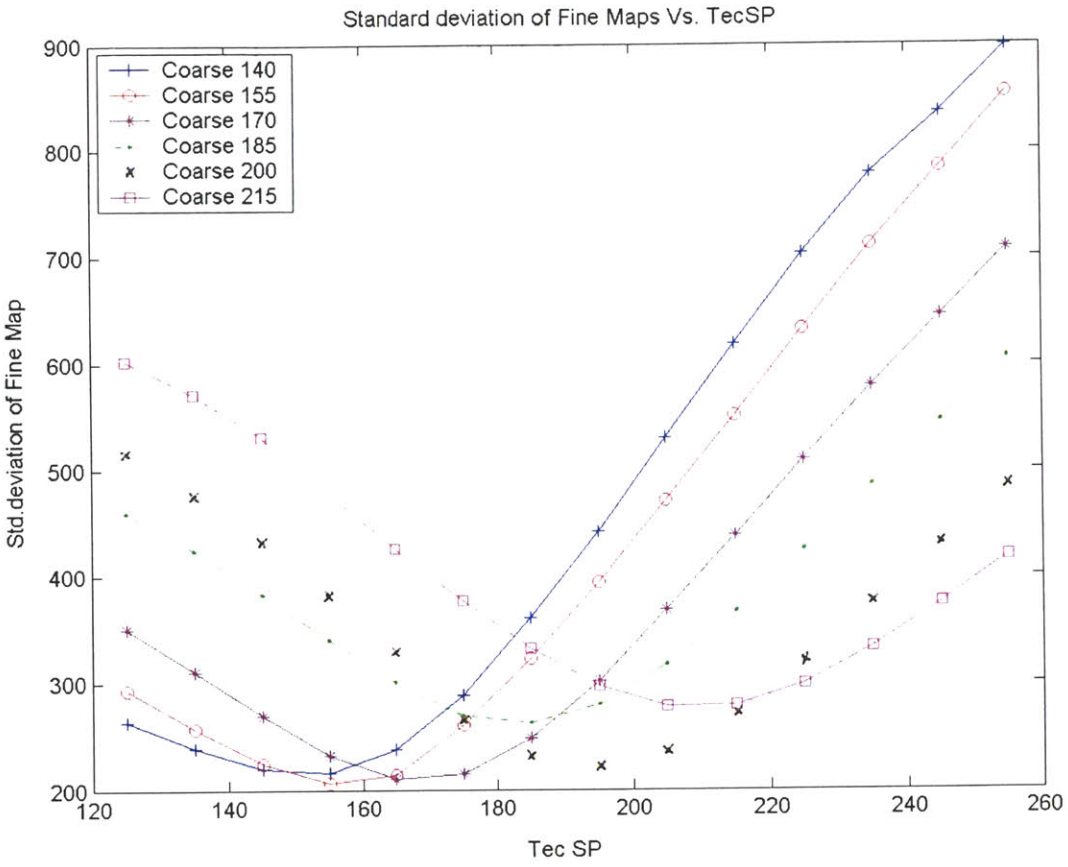


Figure 33: Standard deviation of Fine Maps over temperature (TECSP)

Correction Maps

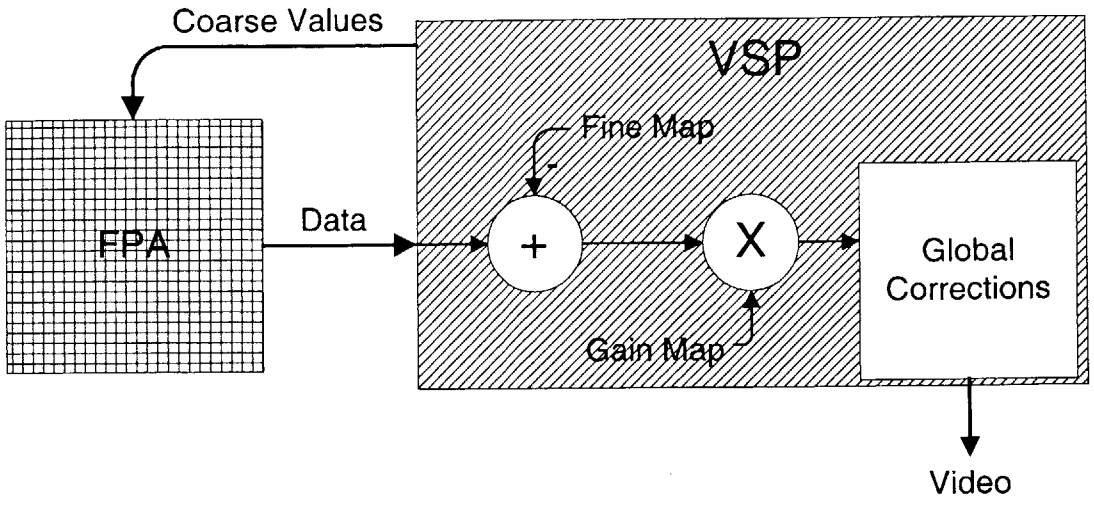


Figure 34: Correction Maps block diagram

In addition to changing *Global Offset*, updates to correction maps are also required for TEC-Less operation over any substantial range of temperatures. There are three possible maps that can be adjusted to offset ambient temperature changes. These are the pixel-by-pixel corrections coarse, fine and gain. In theory, a set of these maps could be calibrated at multiple temperatures and the module could simply apply the set of maps corresponding to the current ambient temperature during use. This approach would likely be effective but would require an enormous amount of memory and time to calibrate sets of maps at many different temperatures.

The approach taken in this investigation is a little more advanced. The coarse and gain were investigated briefly but the main focus was on the update of the fine correction map. The coarse map does not change greatly over a large temperature ranges and is hard to predict, and the fine map is much more exact, making it the best choice to investigate. Therefore a series of three to four coarse maps should be sufficient to cover all operating temperatures. The gain map was not investigated in detail, but if high quality TEC-Less images are to be made, gain maps incorporating ROIC temperature changes may need to be considered.

Coarse Map

As explained earlier, the coarse map correction changes actual resistance values on the ROIC. Each bolometer has an 8-bit digital-to-analog converter (DAC) in series with it. The DAC serves as an adjustable resistor to compensate for bolometer resistance non-uniformity. During calibration, the processor adjusts the value of this resistor until the data value for that bolometer is at a specified target value. This adjustment can be complicated due to the non-linearities in the DAC. These are purposefully designed into the DAC to guarantee that any given bolometer resistance value can be set with the 8-bits. However, the non-linearities make it difficult to predict the correct coarse map value for a given pixel over temperature. To illustrate this problem, Figure 35 shows the coarse value of several pixels across temperature. Four trials were conducted and as the plots show, the coarse value changes are not easily predicted even for the same pixel from trial to trial.

Although the coarse values are not predictable, they do not vary greatly over moderate temperature changes. Again referring to Figure 35 it can be seen that over an approximately 46 degree temperature change most coarse values changed only a few counts. Trials 1, 2, 3 and 4 had worst case deltas of 10, 20, 21 and 21 counts respectively. These somewhat larger counts are most likely due to non-linearities and the actual resistance value did not change as much as the DAC setting would indicate. Most coarse values did not change at all over the entire range. With this data in mind, the decided approach was to try and cover all operational temperatures with a series of discrete coarse maps. Assuming operational temperatures from –40 °C to +85 °C, using three maps requires each to cover a temperature range of 42 °C.

By using a discrete series of maps, the ability to predict coarse values is no longer needed. The coarse map is calibrated at the midpoint of each temperature sub-range and switched in depending on the ambient temperature reading. Video errors associated with coarse map values should be small enough that fine map corrections will be able to compensate adequately.

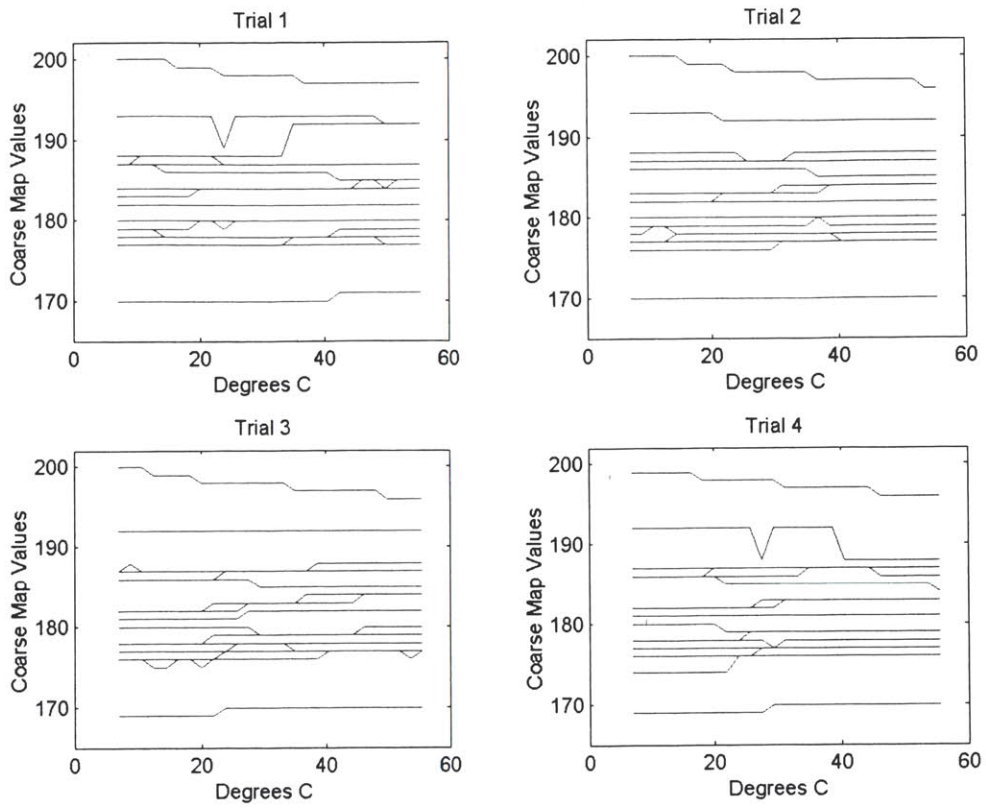


Figure 35: Coarse Map values plotted Vs. temperature for several pixels. Four trials are shown.

Gain Map

The second pixel-by-pixel correction is the gain map. The gain map is determined via PC calibration and typically does not change during normal operation of the module. The purpose of gain is to compensate for differences in response to IR radiation between pixels. The map is calculated by subtracting video frames while looking at a high temperature from frames taken while looking at a low temperature. This difference is the responsivity image and its inverse is proportional to the gain. In practice, the gain map changes little over temperature compared to changes in the fine map, so the gain map is not the focus of any of the experiments.

The gain map is mentioned here simply because it can affect image quality and noise and therefore is mentioned in this report. For all tests the gain was calibrated at the same set point as the coarse map using a high BB temperature of 55°C and a low BB temperature of 25°C. The gain map has been applied in all tests unless stated otherwise.

In order to obtain TEC-Less images comparable to those of TEC controlled situations, gain correction will have to be explored more thoroughly. In particular responsivity differences are temperature dependant and gain corrections dependant on ROIC temperature as well as scene temperature may need to be implemented.

Fine Map

The fine map is a third pixel-by-pixel correction that helps compensate for differences between bolometers. The fine correction compensates for small differences in video values by subtracting the fine map from each video frame. The fine map is an average of several frames taken while looking at a uniform field such as a Black Body or closed shutter. Since this field is uniform, any difference between pixels in the fine map represent inherent differences between pixels. These are then subtracted from video frames to compensate for the inherent differences. The resulting image after fine correction looks much less noisy than previously.

As the ambient temperature changes pixel variances grow, making the image look grainy and noisy. By adjusting the fine map correctly these additional ambient temperature based differences can be subtracted out. The way in which the fine map changes over temperature can be seen in Figure 36. Since the fine map is simply an average of several video frames, the changes shown for the fine map over temperature are exactly representative of the changes of the bolometers over temperature. Notice that the fine map values have the lowest deviation around the calibration and that each pixel is not a linear function of temperature.

Figure 36 below shows the fine map values of several pixels, evenly located across the focal plane, versus temperature. For this test the constant power connection was not implemented. The dashed lines represent coarse and gain map calibration points. Figure 37 shows the standard deviation of the fine map values over temperature. This Figure more clearly shows the low deviation at the coarse calibration temperature and non-linear change from that point.

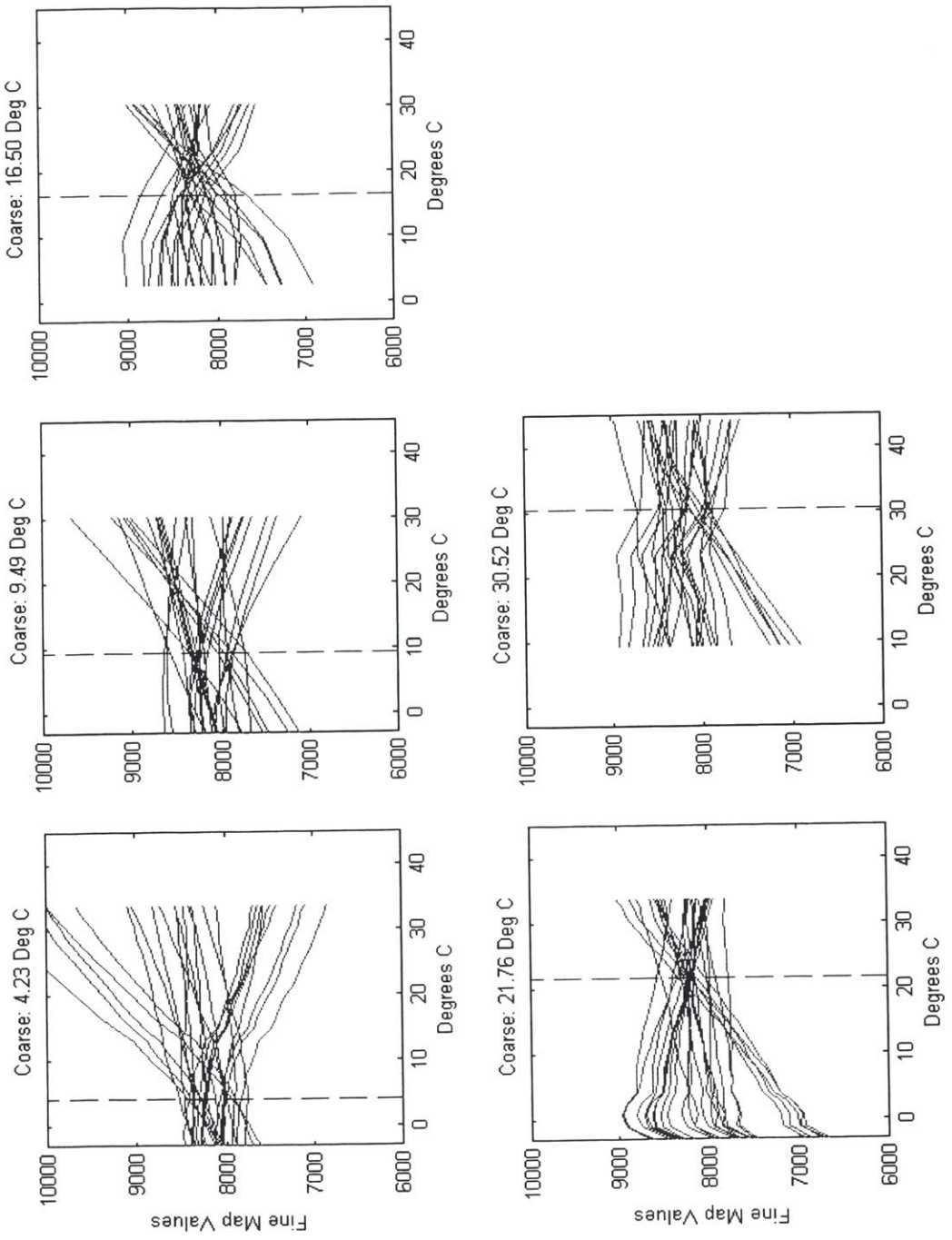


Figure 36: Fine Map values plotted Vs. temperature for several pixels. The dashed lines represent the temperature at which the Coarse Map was calibrated.

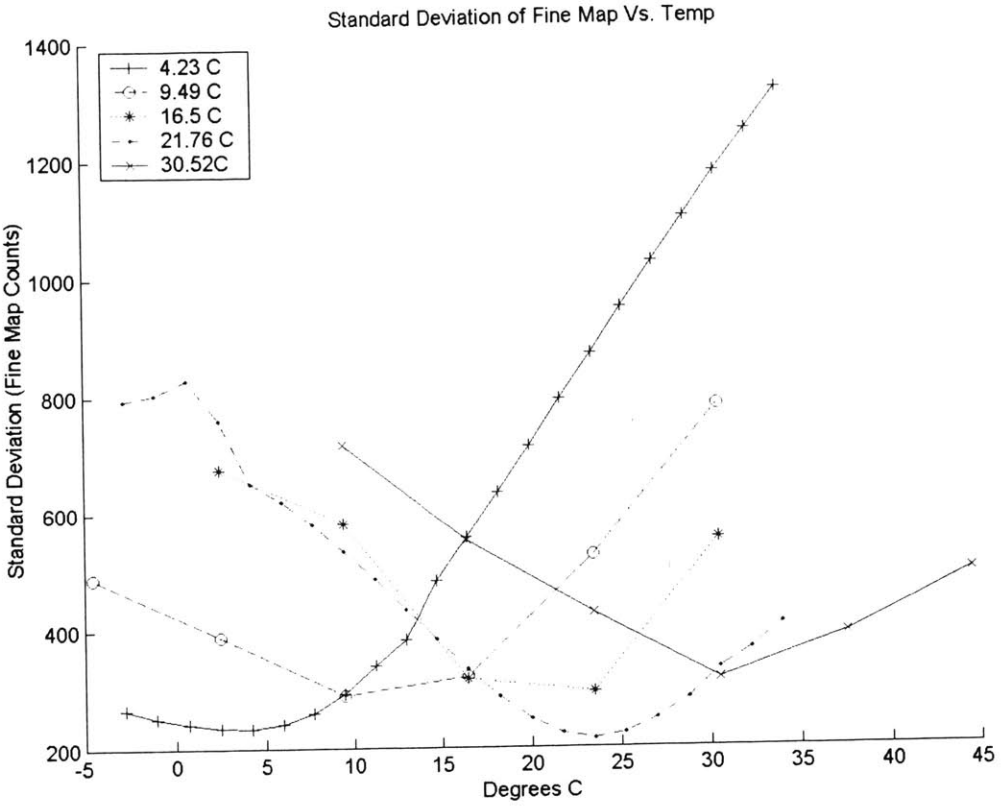


Figure 37: Standard deviation of Fine Maps across temperature. The legend indicates the temperature at which the Coarse Map was calibrated at for each line.

The next set of graphs in Figure 38 contain results from a similar test where the constant power connection was applied and R1/R3 = .4. Again the fine maps have the least deviation at or near the coarse temperature. Notice the even symmetry in the standard deviation graph due to the constant power connection.

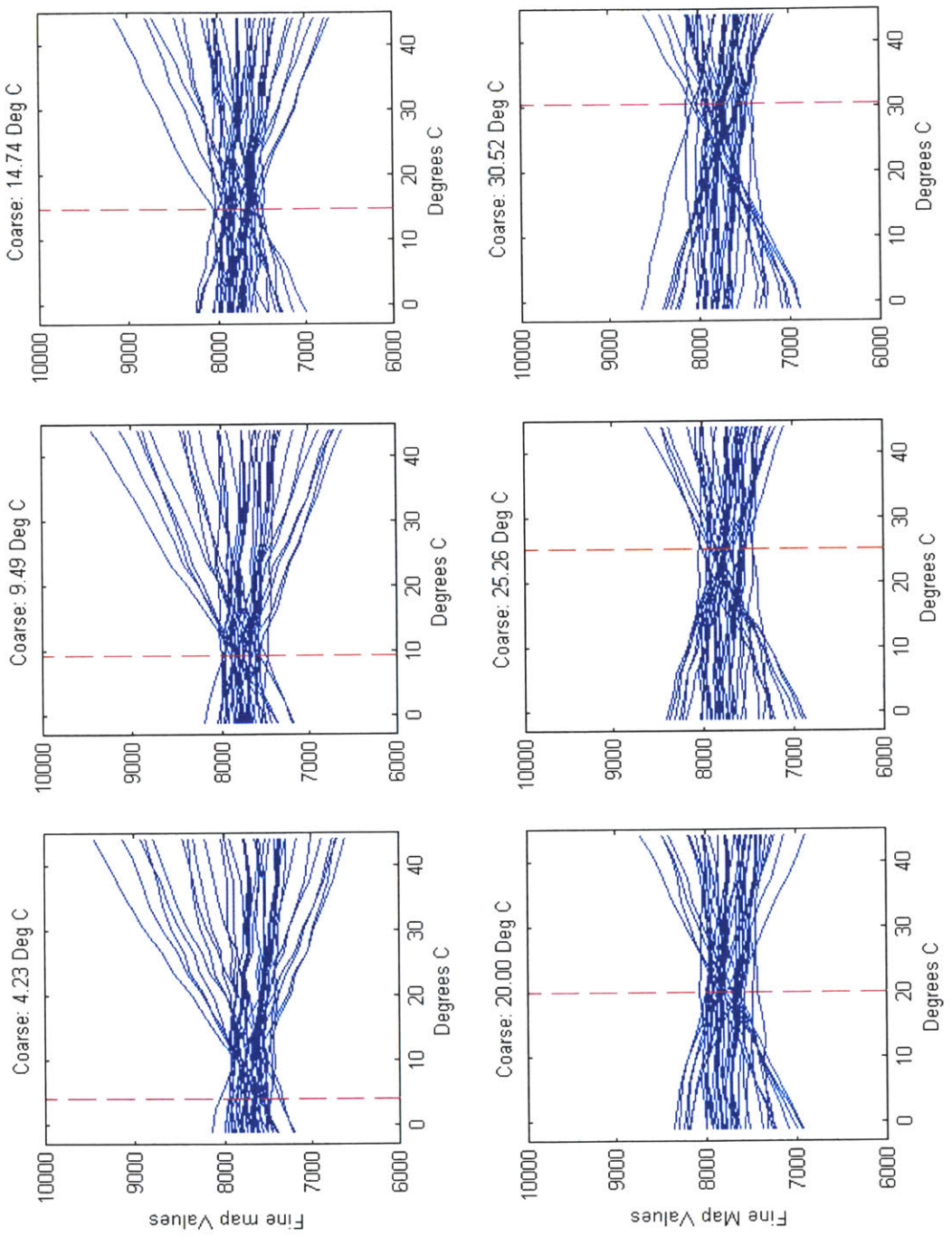


Figure 38: Fine Map values plotted Vs. temperature for several pixels with R1/R3 = .4 The dashed lines represent the temperature at which the Coarse Map was calibrated

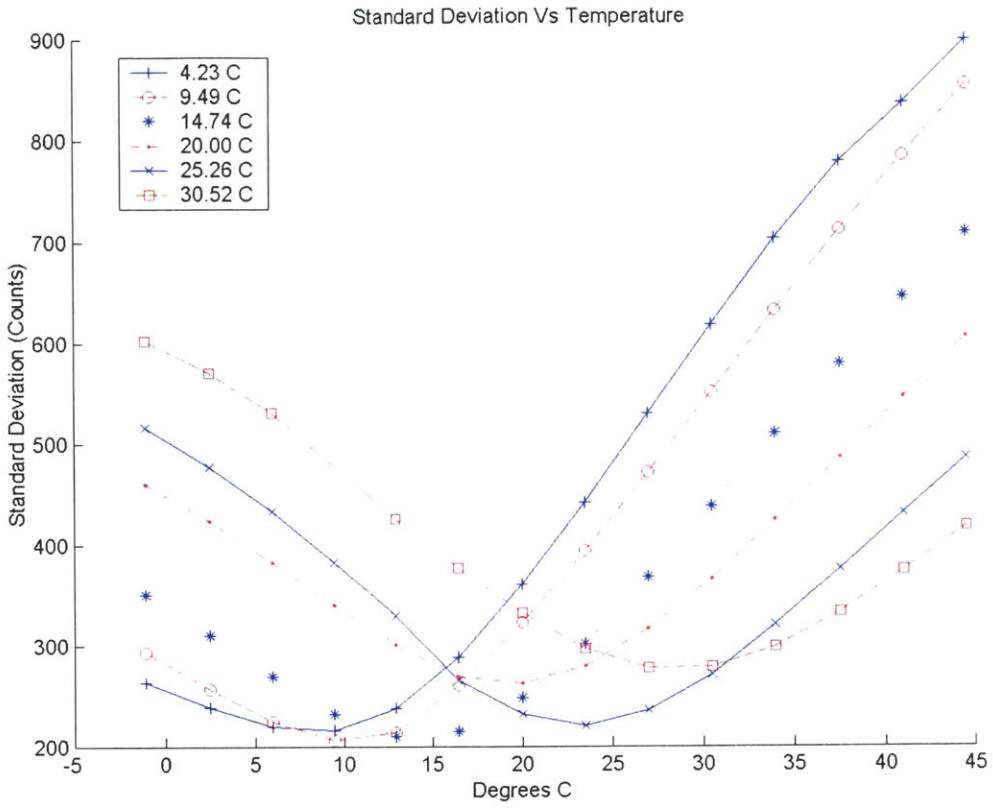


Figure 39: Standard deviation of Fine Maps across temperature with R1/R3 = .4. The legend indicates the temperature at which the Coarse Map was calibrated at for each line.

Spatial Noise

Several tests have been developed by BAE SYSTEMS in order to evaluate the quality of a video image. Despite all efforts, there will always be some inherent noise that appears in every image and the amount of this noise determines if the image is acceptable to the user or not. By turning off the TEC the module becomes susceptible to ambient temperature changes which alter the bolometer resistance. Different resistances across the focal plane can appear as spatial noise in the image, as opposed to temporal noise. Therefore the spatial noise test is the criterion on which the following fine map update algorithms and image improvements are graded.

The spatial noise is measured by averaging 32 frames through time, with the resulting frame titled the S frame. Then the S frame is divided into 5x5 segments and the

standard deviation of each segment computed. The reported spatial noise value is the median value of all 5x5 standard deviations. The goal of the test is to quantify constant differences in pixel values over small areas of the image. Larger gradients, such as shading, are not as noticeable in the image to people and therefore are essentially taken out of the test by using the 5x5 sections.

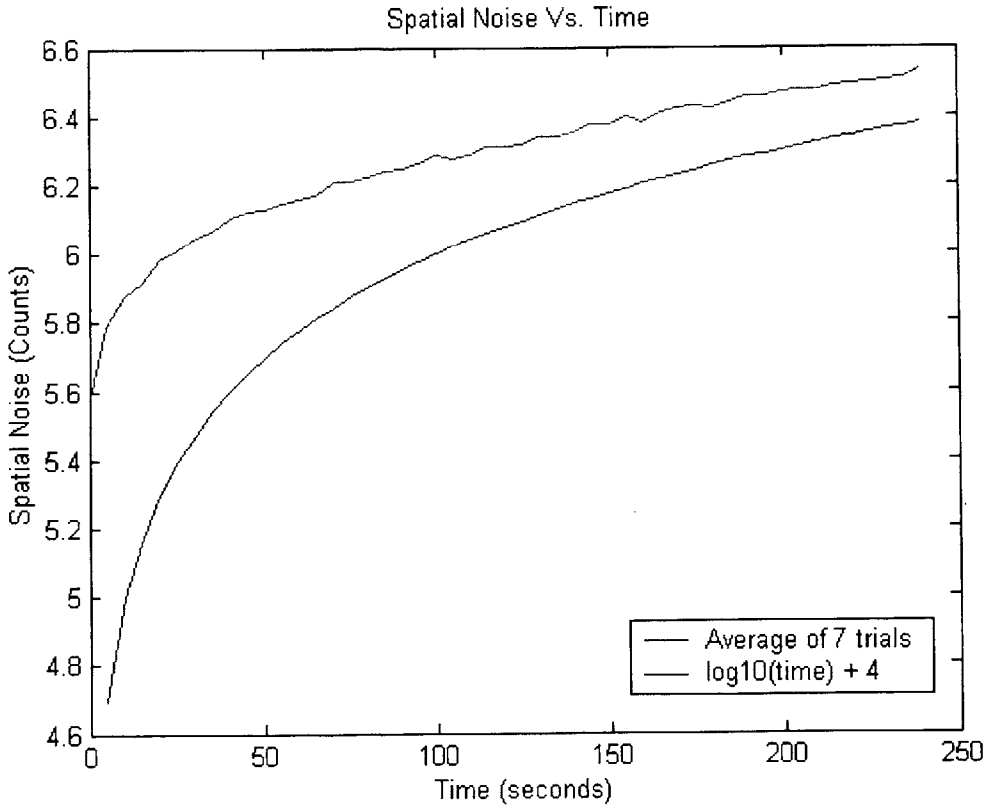


Figure 40: Spatial Noise Vs. Time under normal operation

Figure 40 shows the normal increase in spatial noise with time. The average of 7 trials is shown; the trials were conducted with different coarse maps and shutter operations at several temperatures. Notice the logarithmic increase in spatial noise following a shutter. The shutter operation creates a new fine offset correction map and applies that map to all subsequent frames. This log increase in spatial noise following a shutter operation continues indefinitely and is the main reason that time is an issue in many of the remaining tests. The length of time from shutter until frame collection in

many of the tests is very long. Therefore even under ideal conditions, the spatial noise results will be skewed depending on the elapsed time of the test.

Fine Map Updates

It has been shown that fine map values change when the temperature drifts from the calibration point. The goal of updating the map is to predict the amount of drift realized by each bolometer and adjust the map accordingly.

Local Linearization

The first update approach is local linearization, where the rate of change of each pixel with respect to temperature is calculated during a calibration period and then used to calculate the correct fine map value for each pixel. This method is essentially applying a least squares linear fit to the fine map values over a small range of temperatures and readjusting that fit slightly with data taken at shutter.

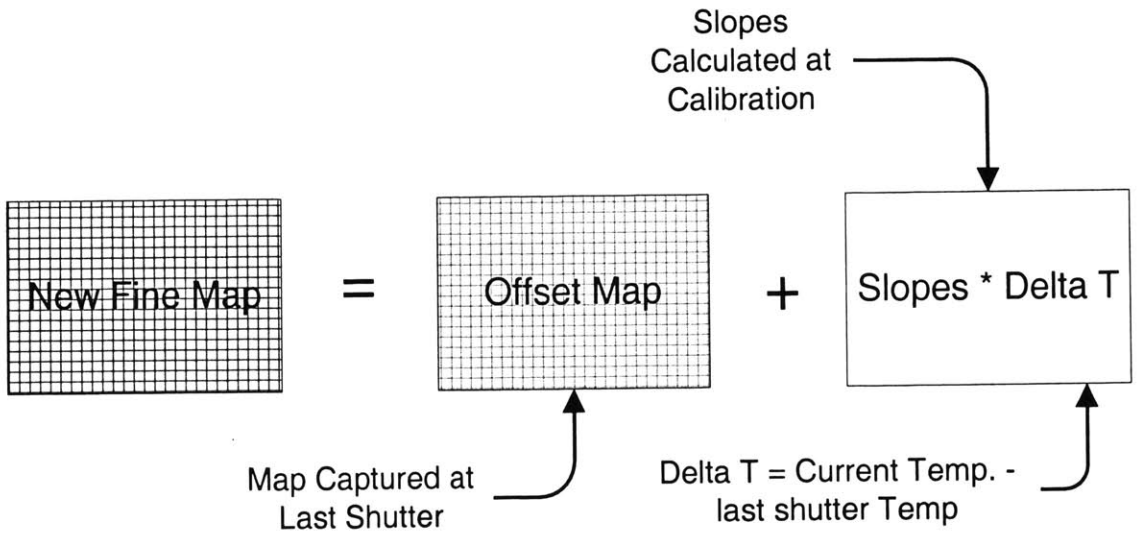


Figure 41: Linear Fine Map Update block diagram

The shutter operation is still essentially the same in the fact that it creates a new fine map by collecting a series of frames with the shutter closed and averages them. This average frame shows the differences between pixels while imaging a uniform scene and under standard operation becomes the fine map until a new one is created at the next shutter. With the local linearization approach the fine offset map is reset at each shutter

in the same manner but is also updated with a temperature related coefficient between shutters. The correction added to each value of the fine map is calculated from the slopes determined at calibration.

The algorithm is shown in block form in Figure 42. The algorithm incorporates the digital *Global Offset* adjustment mentioned earlier as well as the linear update of the fine map. Referring to the right column of Figure 42, The routine is started and the first step is adjusting *Global Offset*. Next, assuming there has not been a shutter routine performed since the last update, the algorithm calculates a new fine map by the routine shown in Figure 41. This new map is then downloaded to the unit and the algorithm is complete. The left column of Figure 42 shows what happens during shutter. Several frames are collected with the shutter closed and averaged. This averaged frame then replaces the offset map in the algorithm (refer to Figure 41).

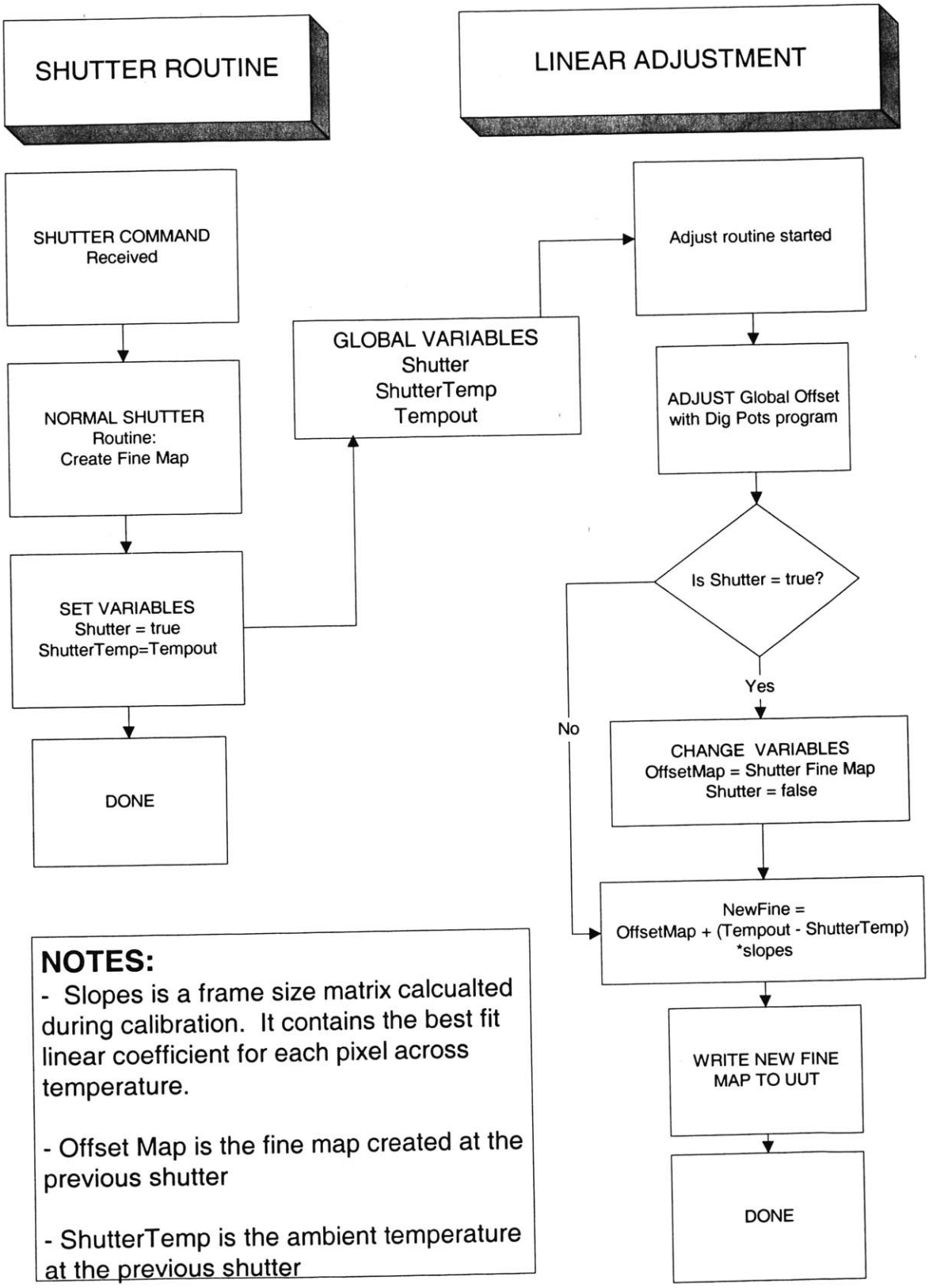


Figure 42: Linear Fine Map Update Algorithm

The linear adjustment is only effective over a limited range of temperatures. The change in fine map values can only be approximated as linear for roughly 5 to 7 degrees. The graphs below show results using the local linearization method. Actual and predicted fine map values are shown across 7°C of change for several coarse calibration points.

Notice that the predicted values are very close to the actual fine map values. A large difference between actual and predicted values would cause a substantial deviation in video frame values, which are plotted in the second set of graphs. The spatial noise of these video frames is plotted last for each test. The spatial noise is high, due in large part to the gain correction map not being applied. Having the gain map applied would have reduced the magnitude of the spatial noise, but it was not applied in order to view more clearly changes to the video output due to the fine map updates.

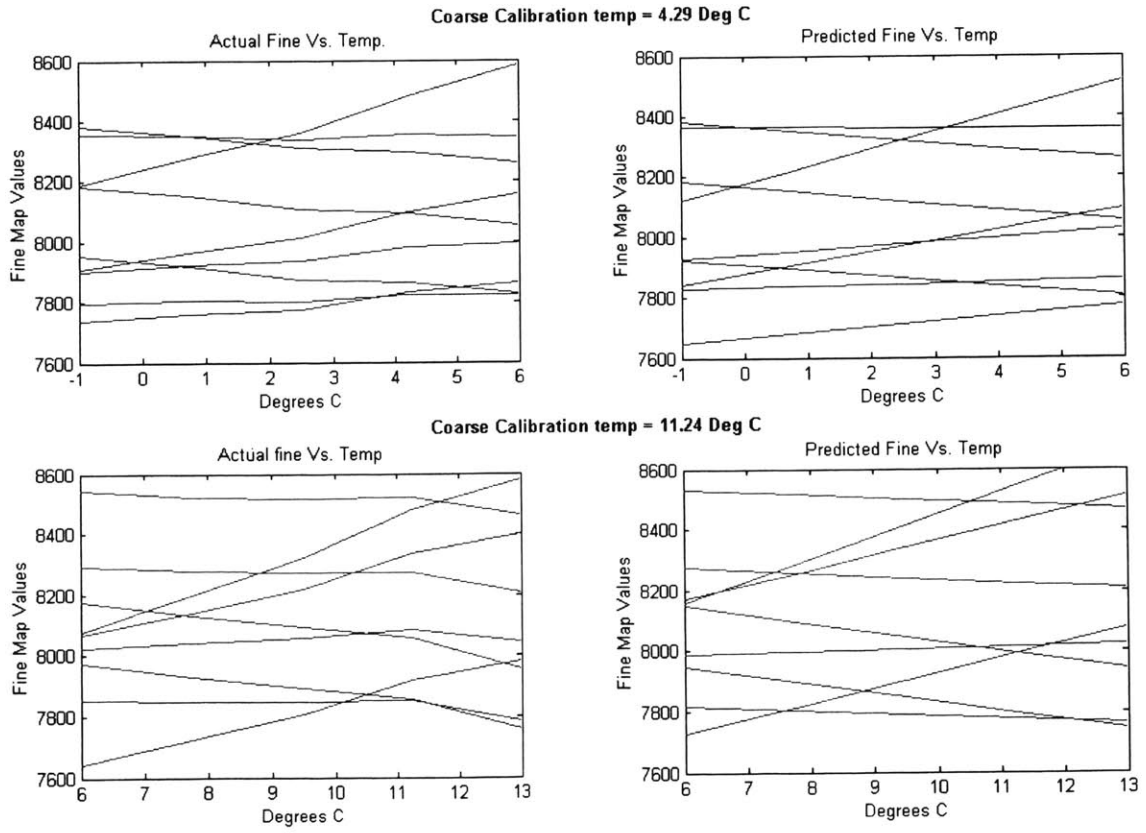


Figure 43: Actual and predicted Fine map values across temperature for several pixels.

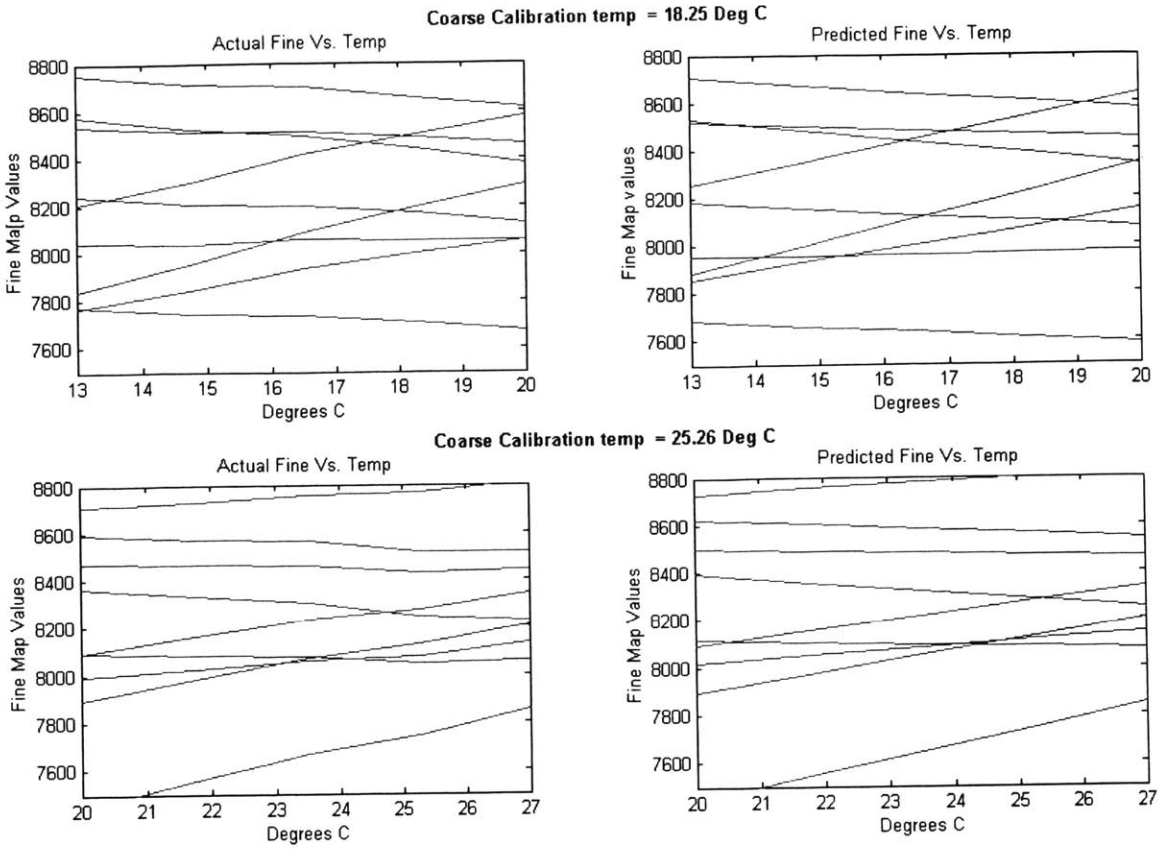


Figure 44: Actual and predicted Fine map values across temperature for several pixels con't.

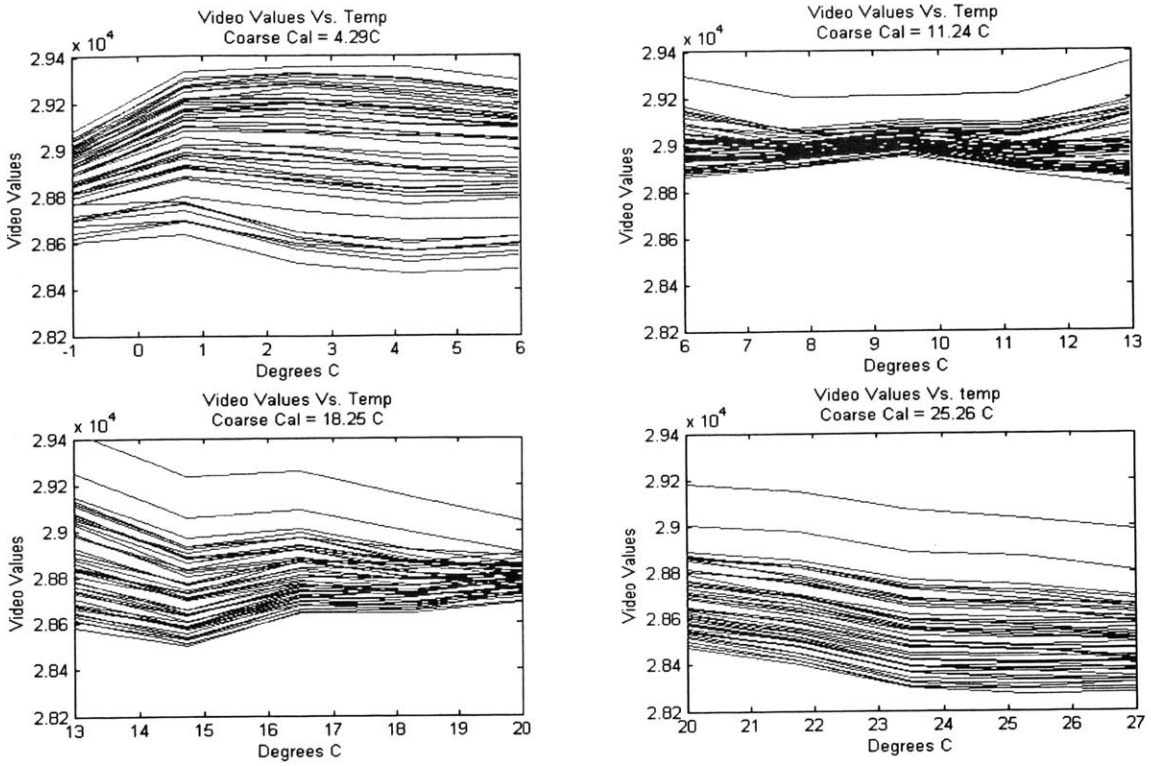


Figure 45: Video Values Vs. Temperature for several pixels with the linear update applied

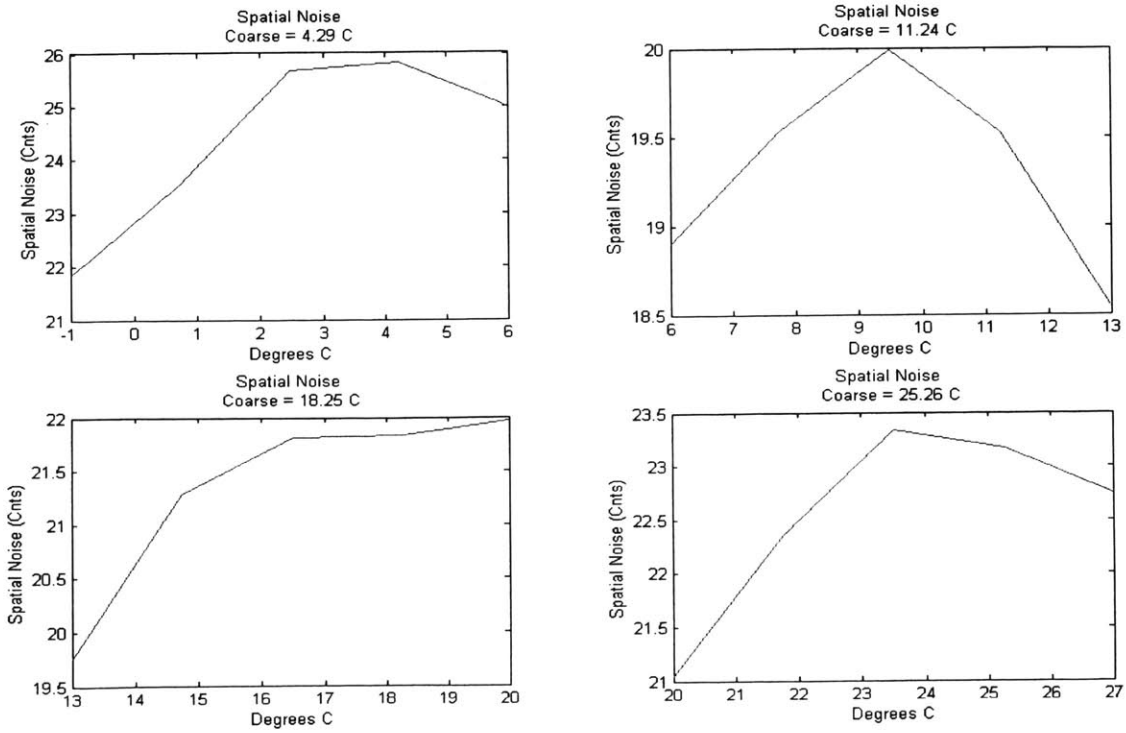


Figure 46: Spatial Noise Vs Temp. with linear update applied, but no gain correction

If the range of the test is extended, the spatial noise increases rapidly near the extremes of the temperature range, as the linear approximation is no longer sufficient. The Figures below show actual and predicted fine map values over temperature with the range equal to 14°C, twice that of the last test. Notice the spatial noise plots and the characteristic curve resulting from the linear fit diverging from the actual fine map values at either extreme. Again the gain map was not applied during these tests, so the spatial noise is higher than it may have been with the gain correction on.

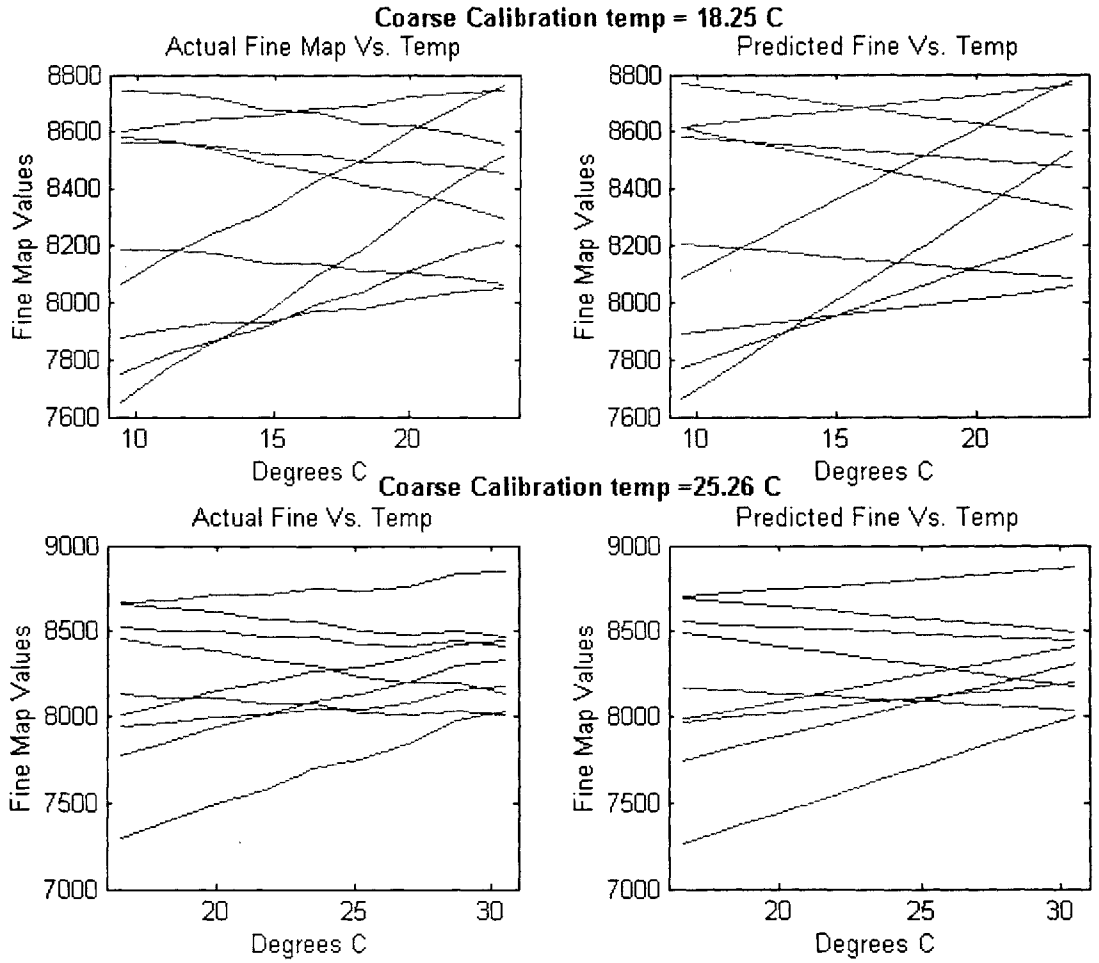


Figure 47: Actual and Predicted Fine Map values for several pixels over extended range of temperatures and two different Coarse values

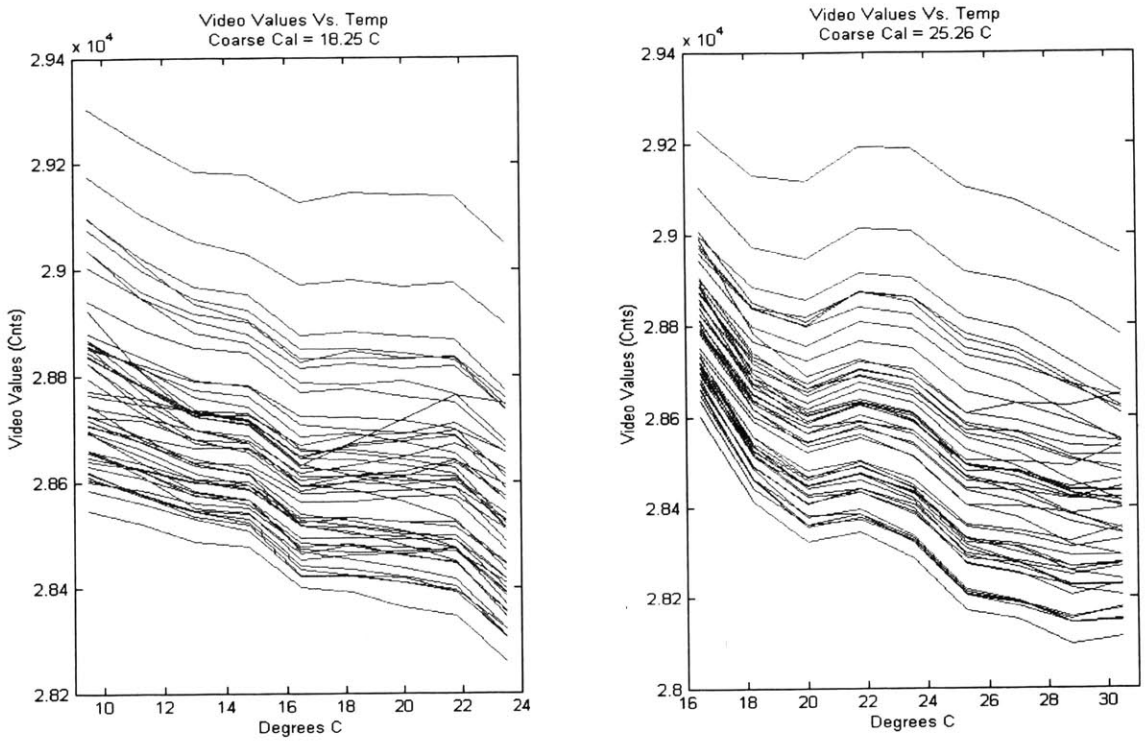


Figure 48: Video values for several pixels across extended range of temperatures

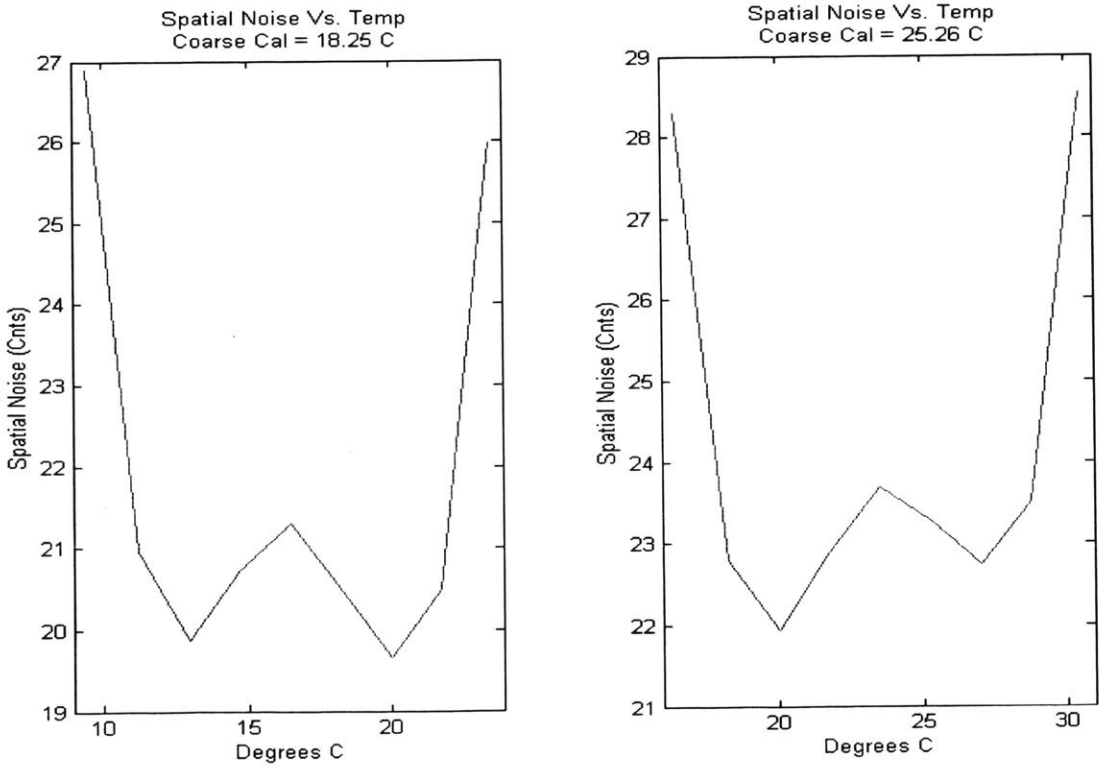


Figure 49: Spatial Noise across extended Temperature range

Cubic Fit

As the data in Figure 38 shows, a linear fit is not sufficient to characterize the changes in fine map values over temperature. A cubic polynomial characterizes the change much more succinctly. Updating the fine offset map with a cubic equation produced very promising results. The cubic fit algorithm is outlined in Figure 50 and is very similar to the linear algorithm except that the new fine map is calculated with a cubic function of t, the current *TempOut* voltage. During each shutter a standard fine map is created as in normal operation, but this map is then used to adjust the cubic algorithm. The map predicted by the algorithm and the map created at shutter are reconciled by adjusting the constant term of the cubic equation so that the predicted map is equal to the shutter map. The cubic adjustment produces low spatial noise counts over ranges of 30 degrees, a much larger range than the linear fit.

Just as with the linear fit, a calibration period is required with the cubic adjustment. The FPA temperature is set at multiple points and fine maps collected. These maps are then used to determine a best fitting cubic polynomial for each pixel. The coefficients are calculated with a least squares fit algorithm, outlined below.

Have the follwing vectors:

$T = [t_1, t_2, t_3, \ldots, t_n]$
where t_x is the tempout voltage at sample x
$Y = [y_1, y_2, y_3, \ldots, y_n]$
where Y is the vector of fine map values for one pixel at T

Want to fit cubic to data:

$$\left.\begin{aligned} D + Ct_1 + Bt_1\text{^}2 + At_1\text{^}3 &= y_1 \\ D + Ct_2 + Bt_2\text{^}2 + At_2\text{^}3 &= y_2 \\ &\vdots \\ D + Ct_n + Bt_n\text{^}2 + At_n\text{^}3 &= y_n \end{aligned}\right\} \text{n equations}$$

Rewrite as:

$$\begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ \vdots & & & \\ 1 & t_n & t_n^2 & t_n^3 \end{bmatrix} \begin{bmatrix} D \\ C \\ B \\ A \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$Z \quad * \quad X \quad = \quad Y$$

Cannot take inverse of Z directly, so first multiply by the transpose:

$$(Z^TZ)*X = Z^TY$$

Then take inverse of (Z^TZ) and multiply both sides giving:

$$X = (Z^TZ)^{-1} * Z^TY$$

The Matlab and C++ implementation of this procedure can be found in Appendix A. This routine was run for each pixel resulting in four 320x240 matrices of coefficients. Obviously if this approach were to be implemented in hardware a sizeable amount of memory is needed to store these maps and more processing time required to compute the new fine map.

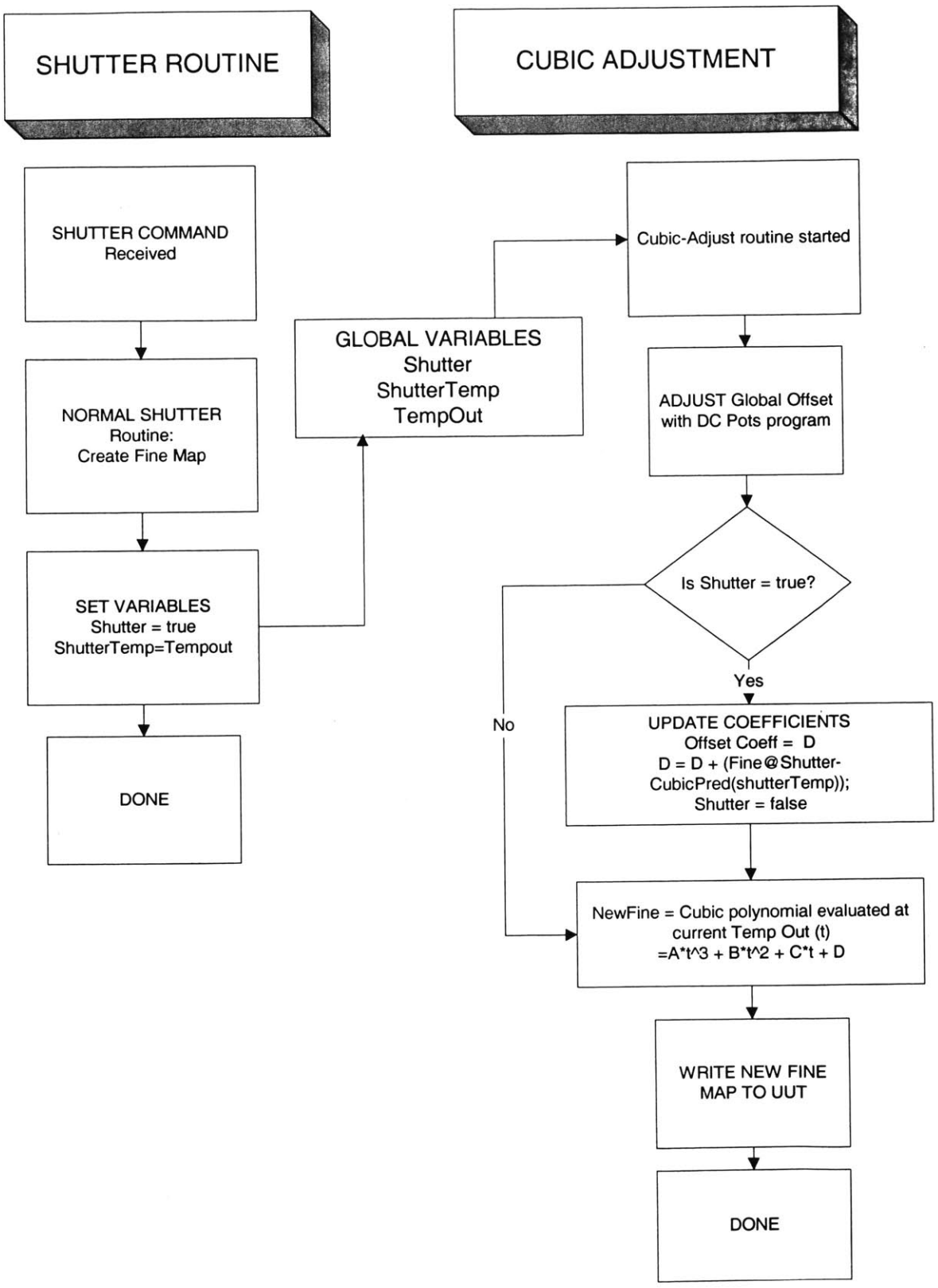


Figure 50: Cubic Fine Map Update algorithm

Test Results

The following tests show results for the cubic fit over three large temperature ranges. The total range is –15°C to 85°C. These temperatures were chosen because some of the electronics in the module began to show abnormal behavior below and above this temperature range. Dividing into three equal ranges gives a low range of –15°C to +18°C, a mid-range of 18 to 51°C and a high range of 51 to 85°C. Within each range a coarse and gain map were calibrated as well as cubic coefficients calculated.

Tests were conducted with shutter routines completed at different ambient temperatures. This is important because both the cubic and linear fits are adjusted to match the fine map calculated during shutter with the one predicted for that temperature. For example, in the low range there are three different shutter temperatures listed, one of them being 5°C. This means that the FPA temperature was set to 5°C and a shutter routine completed. This is the only shutter routine for the entire trial, and the map created during shutter is used to adjust the constant term of the cubic equations or alternatively the offset if a linear fit is being used. The temperature is then varied across the low range of temperatures and the fine map adjusted according to the cubic or linear equations. The test is then repeated for the two other shutter temperatures, as different shutter points change the fits slightly.

Tests were conducted both with the global corrections turned off and with them on. The global corrections, or Automatic Error Correction (AEC), apply several corrections including auto gain and auto equalization. Auto equalization uses non-linear histogram equalization to make more pixels viewable. The data shows that in most instances AEC reduced the spatial noise present in the image. The linear fit results are also graphed for comparison. Each line is the average of 2-3 trials.

Test Problems

There were several problems encountered with the tests. First, the amount of time taken for each test is an issue. Updates to the fine map were conducted on average every 2 degrees over each range, corresponding to approximately 17 updates. Each test started with a shutter routine at the given shutter temperature, and then the temperature was

varied across the range again with an update every 2 degrees. Each of these updates takes a considerable amount of time. First, approximately 20 seconds is needed for the FPA to reach equilibrium after the TEC has changed temperatures. Second, the time for the GO adjustment is variable but on average takes several iterations of the algorithm each taking several seconds. Third, the fine map adjustment calculation is very quick on the PC but downloading the new map to the unit takes at least 35 seconds. Lastly there are multiple commands that are sent from the PC to the unit all of which take a small amount of time. Combining all these times results in an average update lasting 110 seconds. This can lead to test times around 32 minutes, and sometimes as long as 40 minutes. Clearly waiting 32 minutes from the time of shutter to the collecting of frames is going to add to the spatial noise as shown earlier.

A second problem was the calibration of the unit within each range. A coarse and gain map were calibrated for the unit as well as a set of coefficients for both the linear and cubic adjustments in each range. However, in order to create these coefficient matrices many fine maps across temperature have to be collected. This procedure also took a long time and often needed to be repeated. Unless otherwise stated, the results below use one set of coefficients for all the trials. However, in the low and mid-range more than one coefficient set was used. Averages of trials are plotted and these averages may include trials from different coefficient sets.

Low Range

LOW RANGE			
Temp. Range	-15C	+18C	
TECSP	86	180	
Shutter Temps	-6.5C	5C	14.5C
Coarse	+5C		
Chamber Temp	-20C		

Table 2: Low Range test values

For the low range test the chamber temperature was set at –20°C, which allowed the TEC to cool the FPA to –15°C. The FPA was still used to change the ROIC temperature over the entire range as it was much quicker than letting the chamber temperature settle at each test point. Data was collected every 2.1°C over the entire range, with two trials being conducted at each shutter temperature and then averaged for

both the cubic and linear fits. All the linear trials were conducted with a single slope matrix, while the cubic plots are averages of 2 trials each with a separate set of cubic coefficients being used. One coarse and gain map was used for all the low range tests and they were calibrated at 5°C.

Since the chamber was used for this test the scene being imaged is no longer provided by a BB but is a uniform scene at the same temperature as the chamber, in this instance –20°C.

Notice that the spatial noise is relatively flat across temperature for the cubic fit even if it is large without the AEC corrections. The results with the AEC corrections applied are particularly promising as the spatial noise approaches levels of normal TEC operation of 4-6 counts. The goal of course is to keep spatial noise at or below the TEC levels for all temperatures.

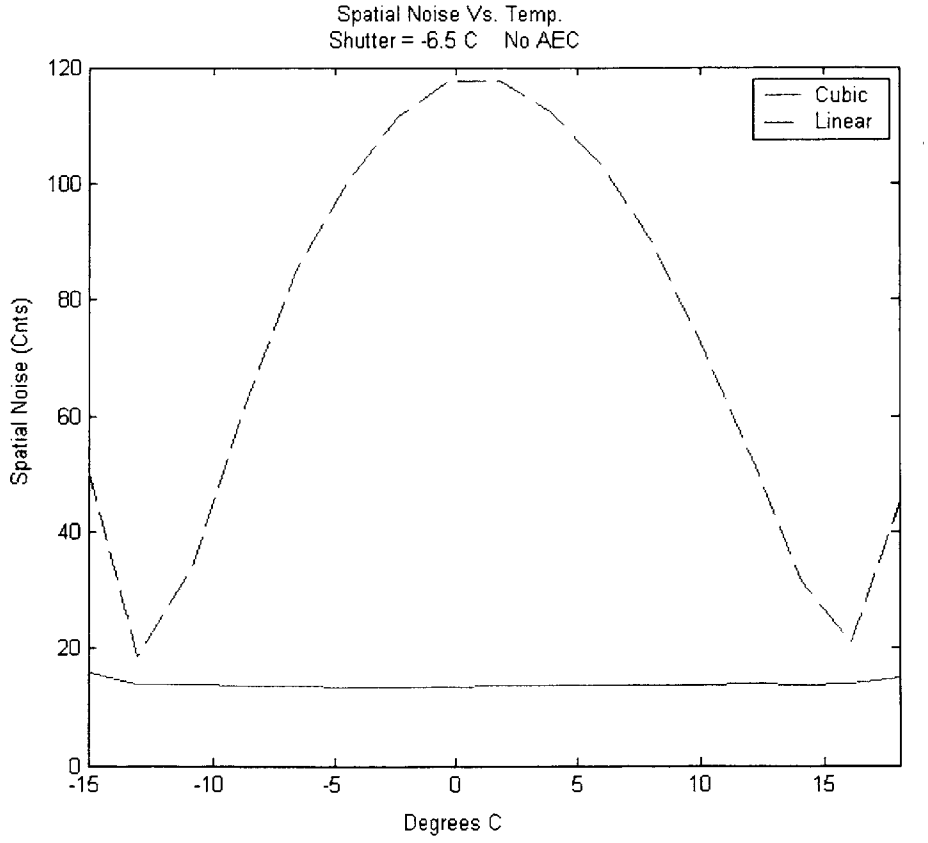


Figure 51: Spatial Noise Vs. Temperature for linear and cubic adjustments without AEC. Shutter = -6.5 C

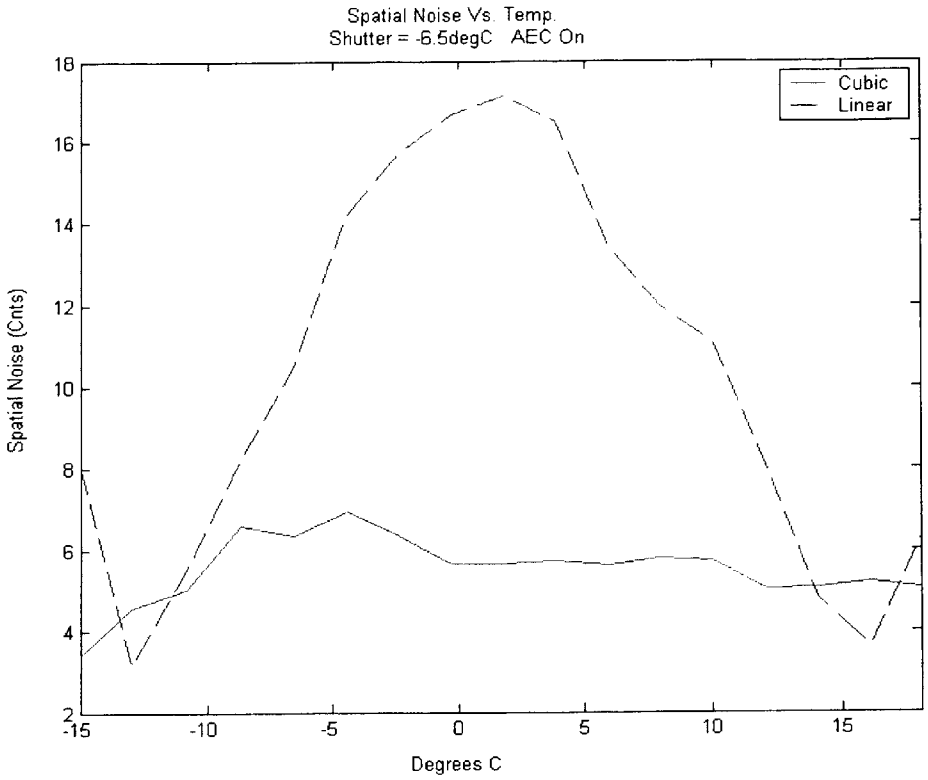


Figure 52: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = -6.5 C

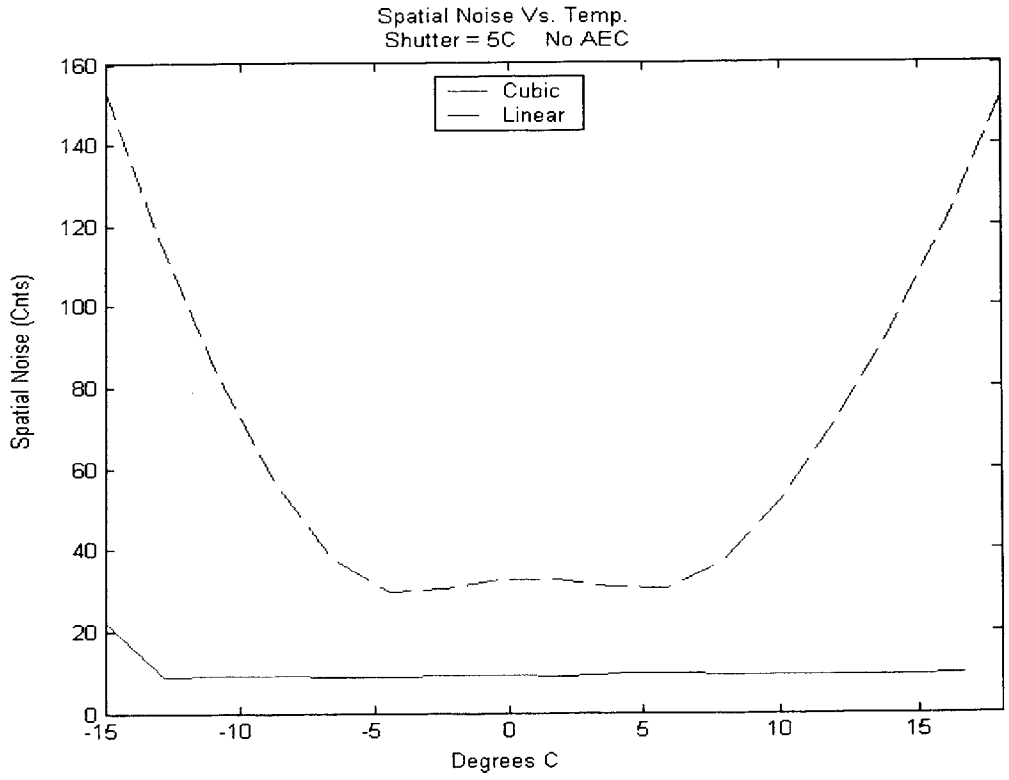


Figure 53: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 5 C

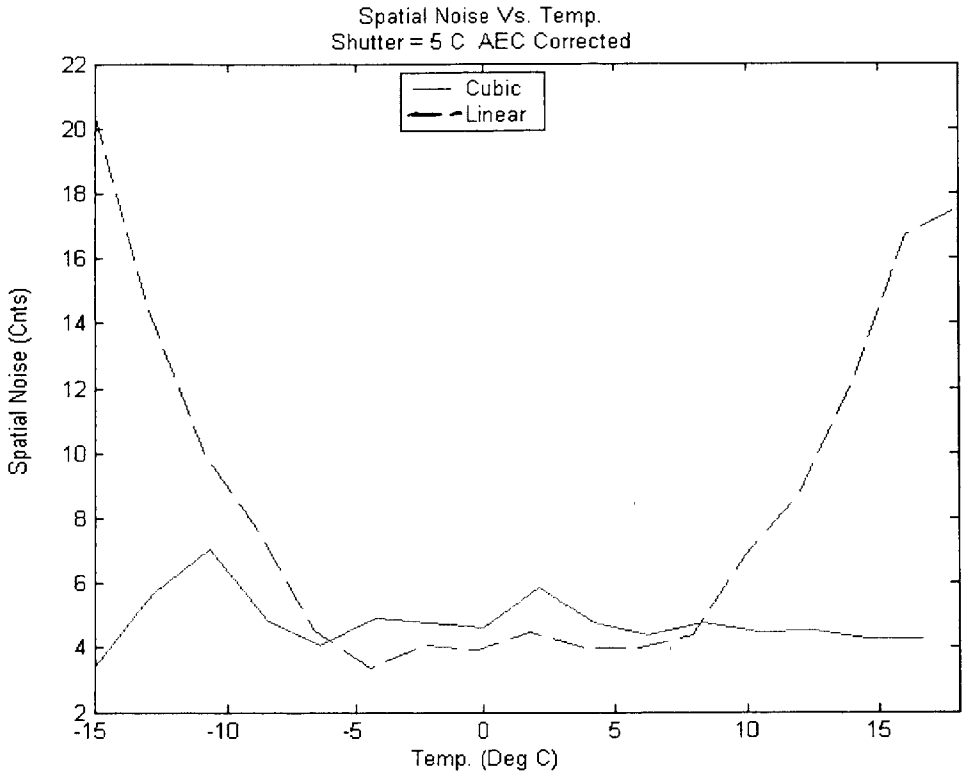


Figure 54: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 5 C

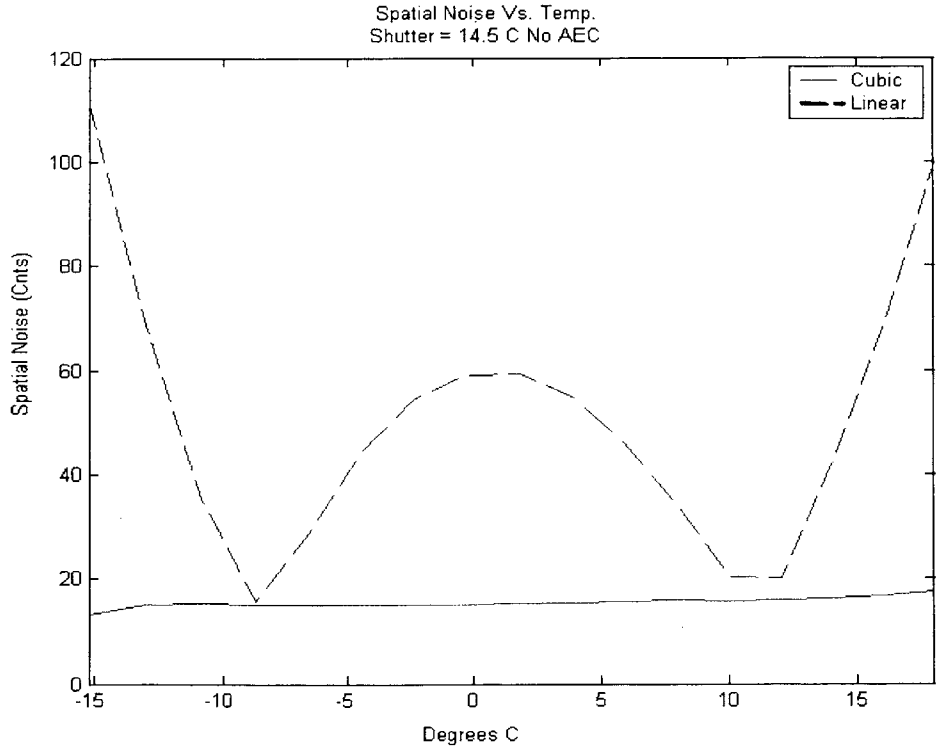


Figure 55: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 14.5 C

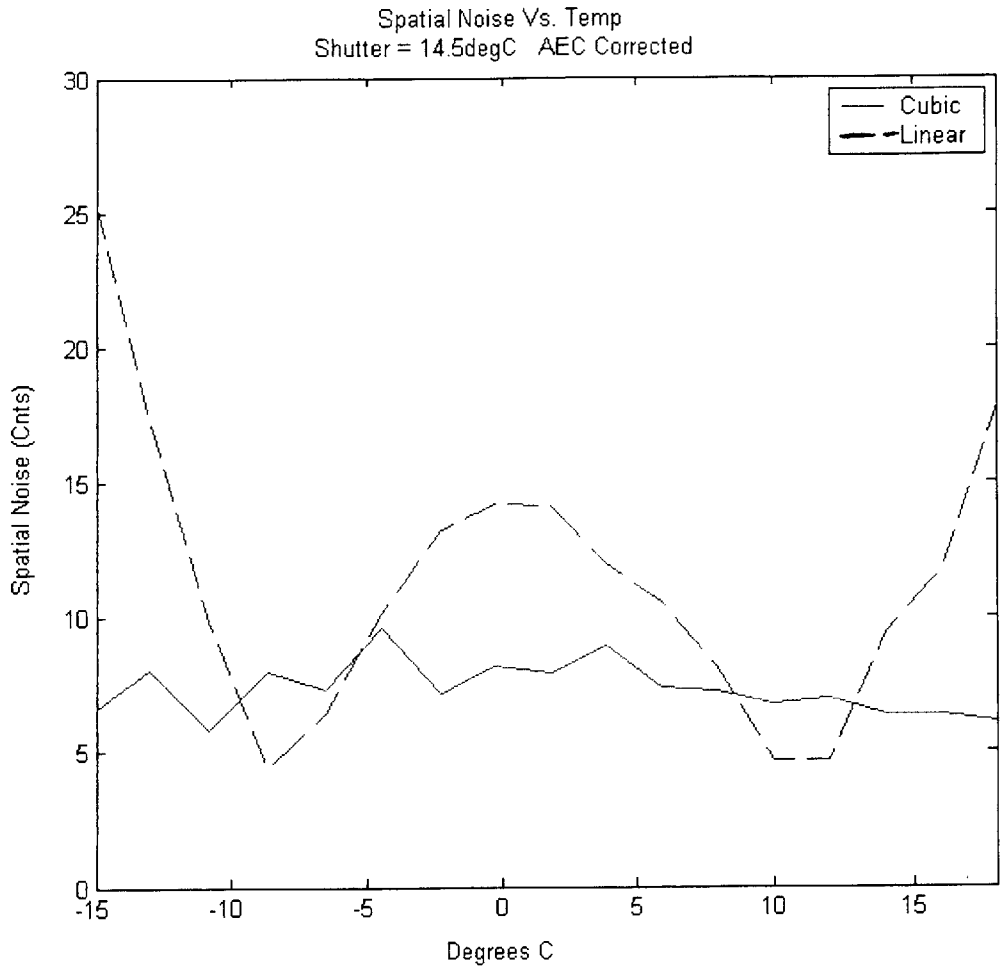


Figure 56: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 14.5 C

Mid-Range

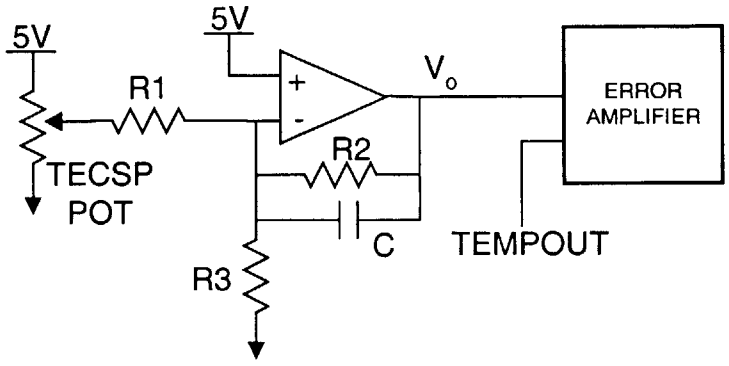
MID RANGE			
Temp range	+18C	+51C	
TECSP	122	190	
Shutter Temps	29C	38C	43C
Coarse	38C		
Chamber Temp	35C		

Table 3: Mid-Range test values

The middle range spanned the temperatures from 18 to 51 degrees C, with shutter operations at 29, 38 and 43 degrees.

Changing the TECSP

A minor problem was encountered when using the TEC to change the FPA temperature. The theoretical limit for the current TEC setup is 44.5°C due to the TECSP circuit, which is shown in Figure 57. With the potentiometer set at the highest setting the voltage going into the error amplifier is 6.505V. The Tempout voltage corresponding to 51°C is 6.359V (Tempout goes down for increases in temperature). Therefore the TECSP circuit had to be modified to allow the TEC to go to higher temperatures, both for these tests and the high range tests. Referring to Figure 57, by adjusting the ratios of R2 to R1 and R2 to R3 the range of Vo could be increased allowing for a larger overall range of voltage values. Increasing this range also meant larger individual changes in temperature given a single step in the TEC pot, but this was not a concern for these tests. R1 and R3 were changed to allow an overall set point voltage of 5.65V to 8.51V. This gives a theoretical temperature range of -42.98°C to 83.35°C, allowing the TEC to reach all temperatures needed for the remaining tests.



$$\frac{TEC - V_-}{R_1} - \frac{V_-}{R_3} + \frac{V_o - V_-}{R_2} + \frac{V_o - V_-}{1/Cs} = 0$$

$$V_o = V_-\left(\frac{R_2}{R_1} + \frac{R_2}{R_3} + 1\right) - \frac{R_2}{R_1}TEC$$

With Initial Values gives:
Vo = 8.5115 - .4013 (TEC)
or a range of 6.505V to 8.5115V corresponding to a theoretical temperature range of -44.8°C to 44.54°C

Figure 57: TECSP circuit and derivation

The results of the mid-range tests are very similar to the low range, with the cubic adjustment keeping a near constant spatial noise value across temperature. With the exception of the 29°C test, the graphed results are averages of 2 trials with both the cubic and linear using 2 different sets of coefficients. The 29°C test results are the results of a single trial for each method.

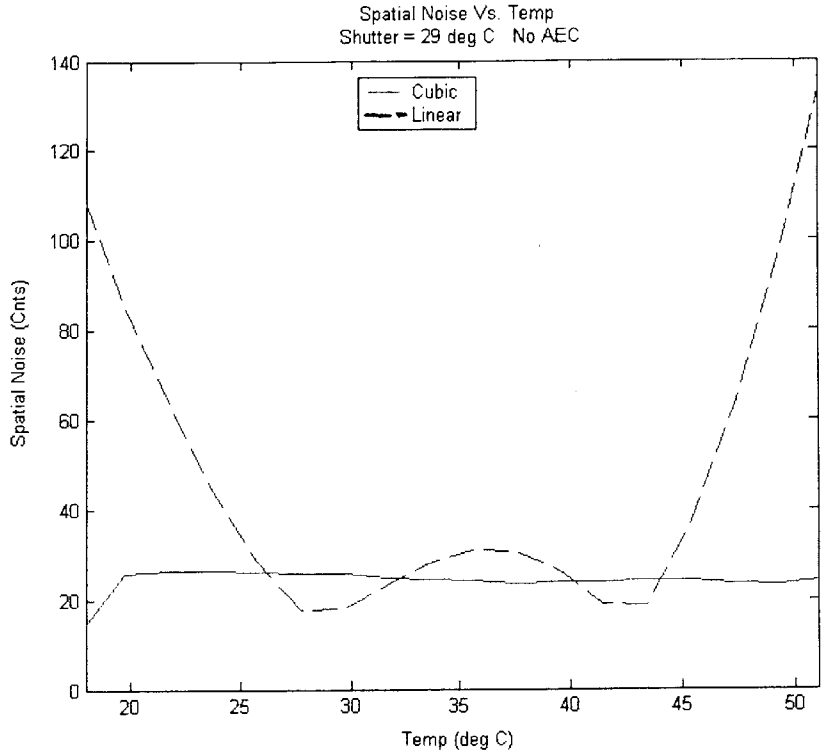


Figure 58: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 29 C

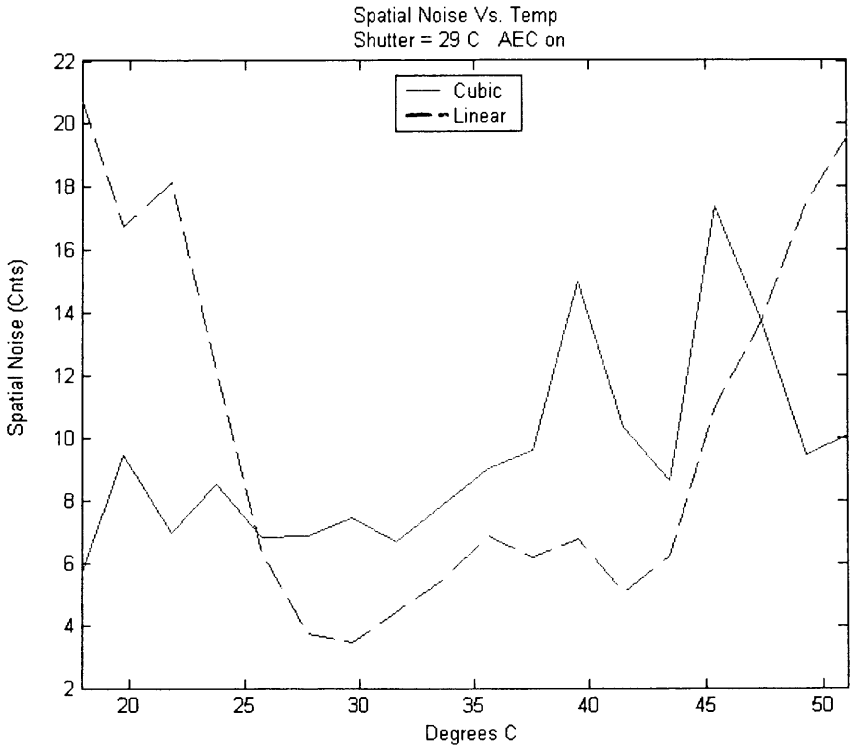


Figure 59: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 29 C

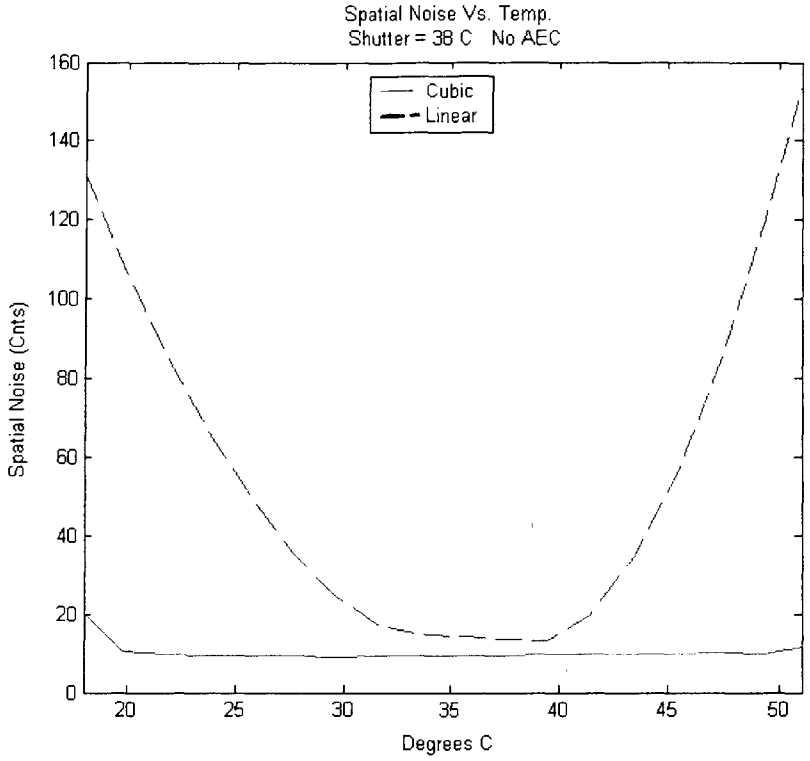


Figure 60: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 38C

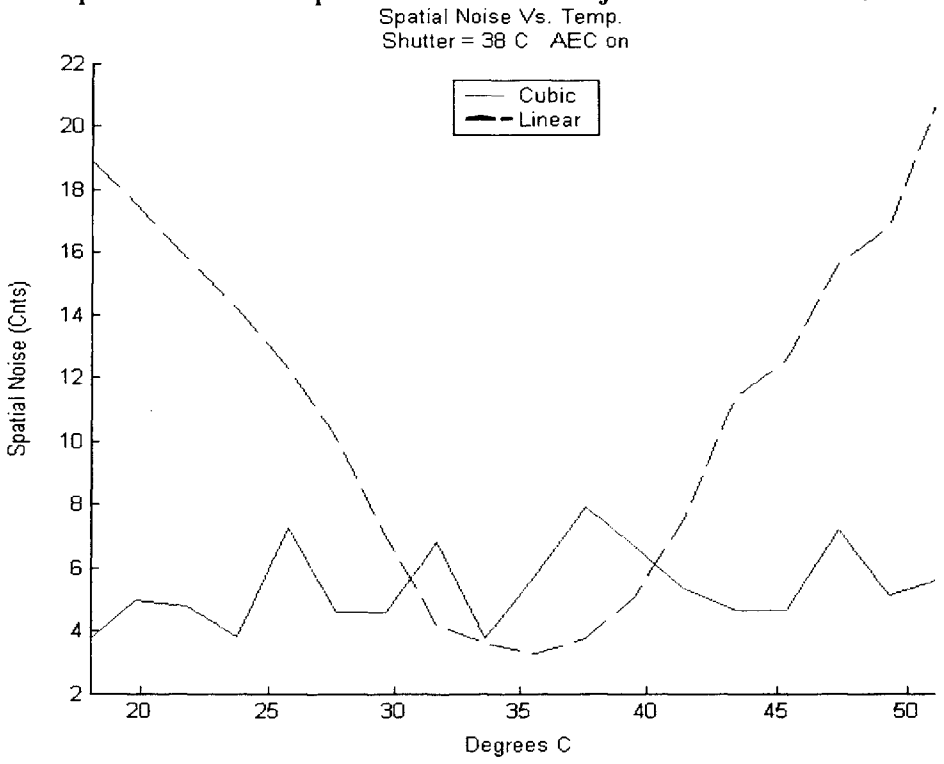


Figure 61: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 38C

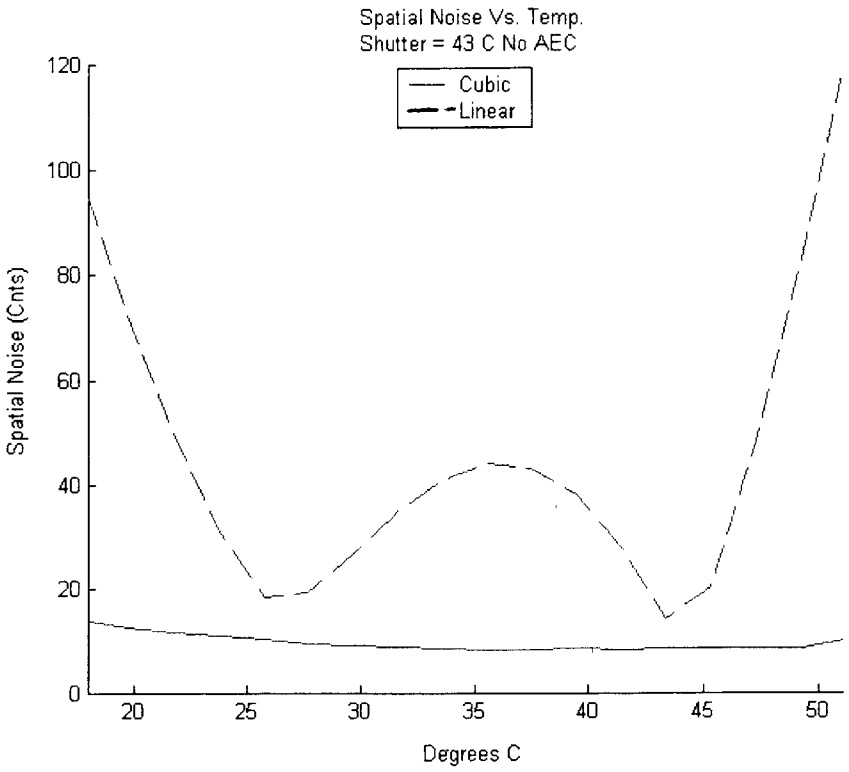


Figure 62: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 43C

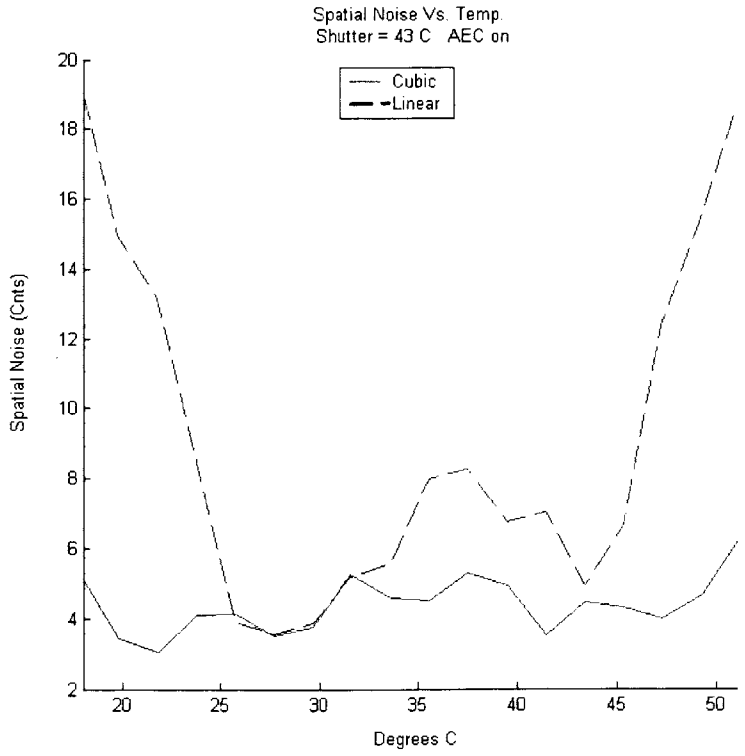


Figure 63: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 43C

High Range

HIGH RANGE			
Temp range	+51C	+84C	
TECSP	189	255	
Shutter Temps	56C	71C	77C
Coarse	71C		
Chamber Temp	55C		

Table 4: High-Range test values

The spatial noise values are the least linear in the high range. Modules often have a difficult time operating at very high temperatures and therefore the cubic adjustment works least well in this environment. Again three shutter temperatures were tested, being 56, 71 and 77 degrees. The coarse and gain were calibrated at 71degrees. The results show that the cubic adjustment is much better than the linear approach over this large range of temperatures, but this is no surprise after seeing the previous graphs. The chamber temp was kept at a fairly low temperature of 55°C compared to the temperature of the FPA. Increasing the chamber and therefore the scene temperature to much greater temperatures may possibly have an adverse effect on the cubic adjustment.

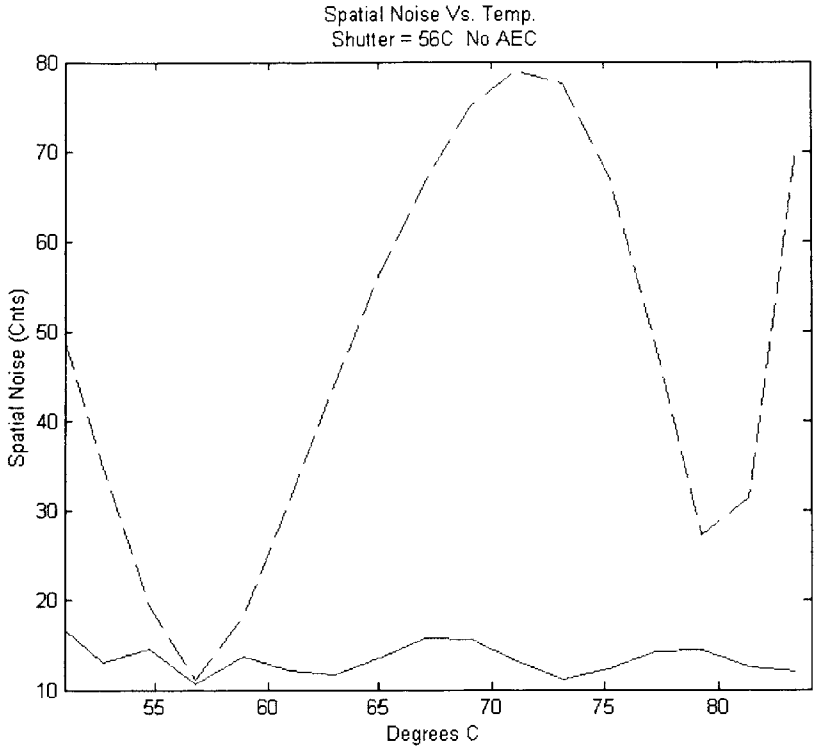


Figure 64: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 56C

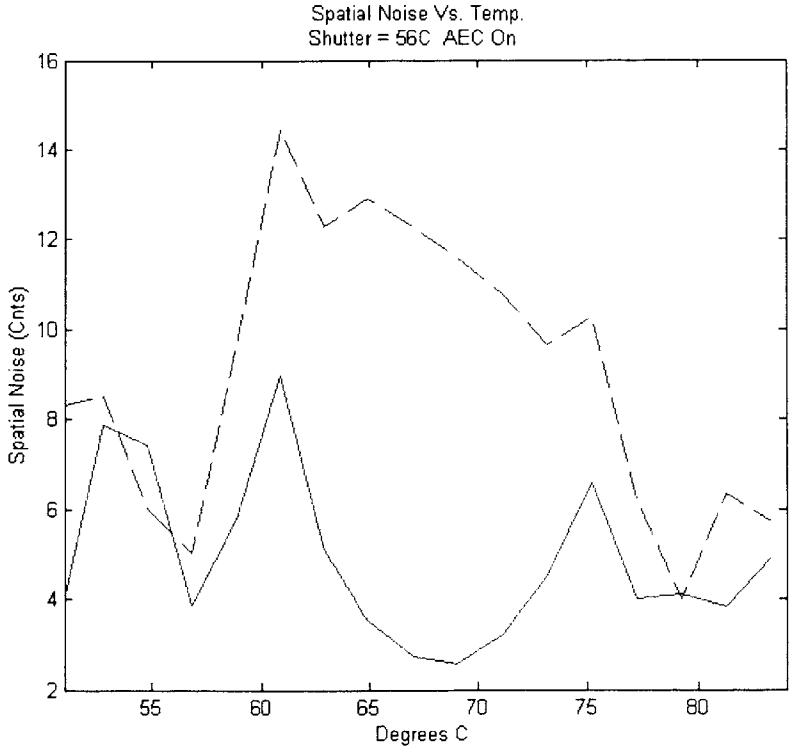


Figure 65: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 56C

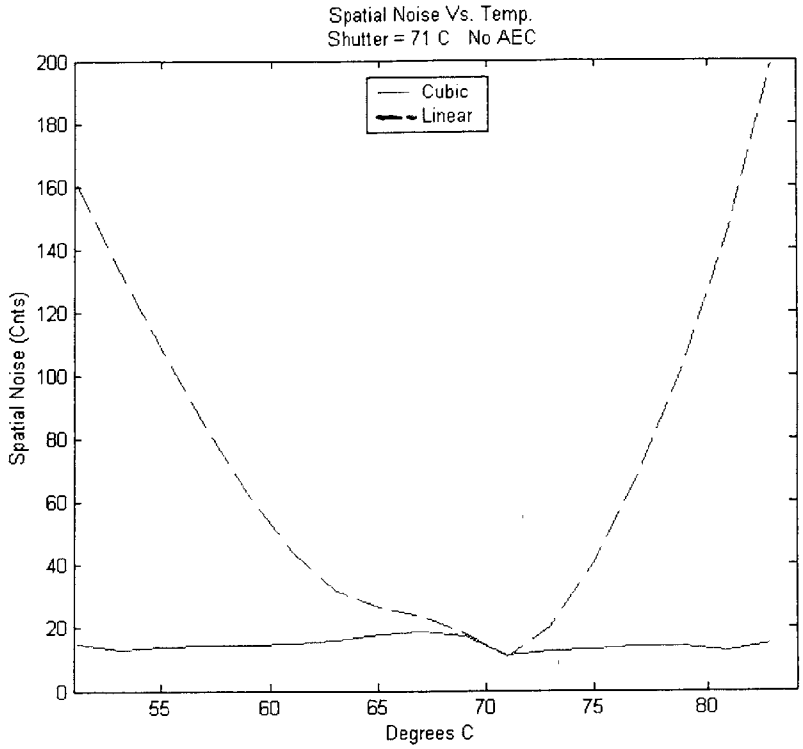


Figure 66: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 71C

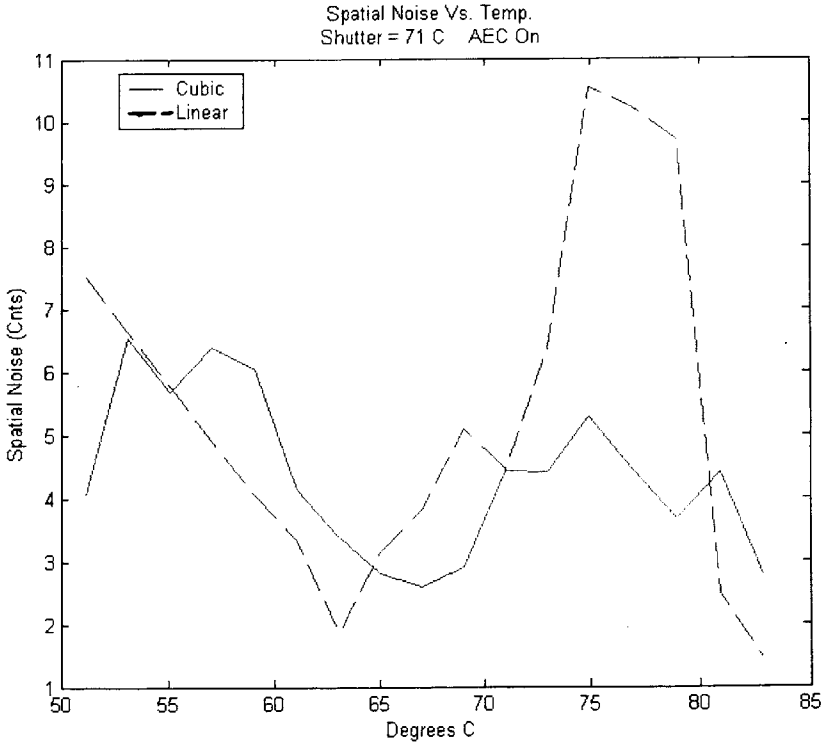


Figure 67: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 71C

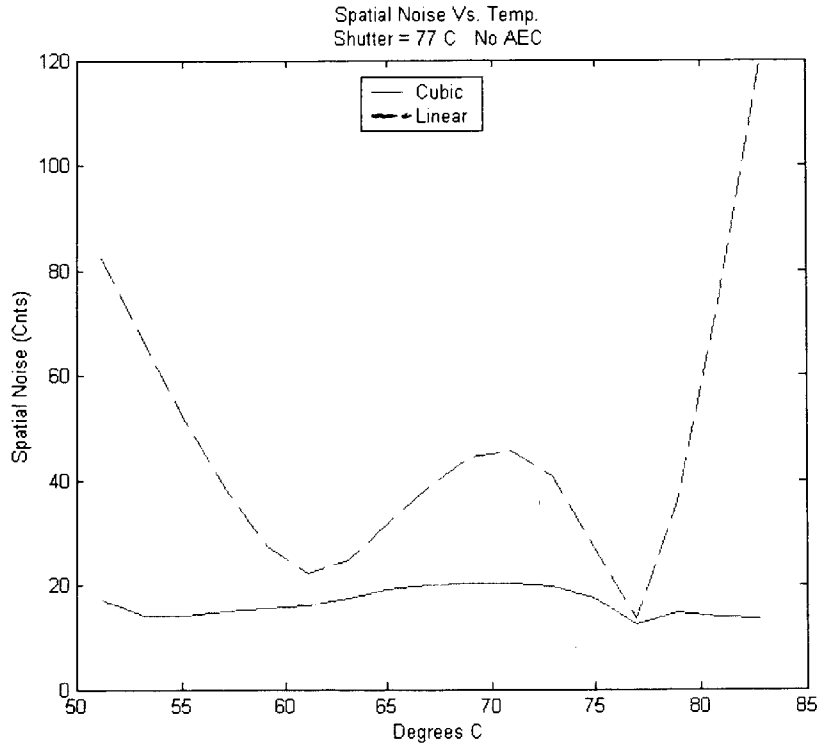


Figure 68: Spatial Noise Vs. Temp. for linear and cubic adjustments without AEC. Shutter = 77C

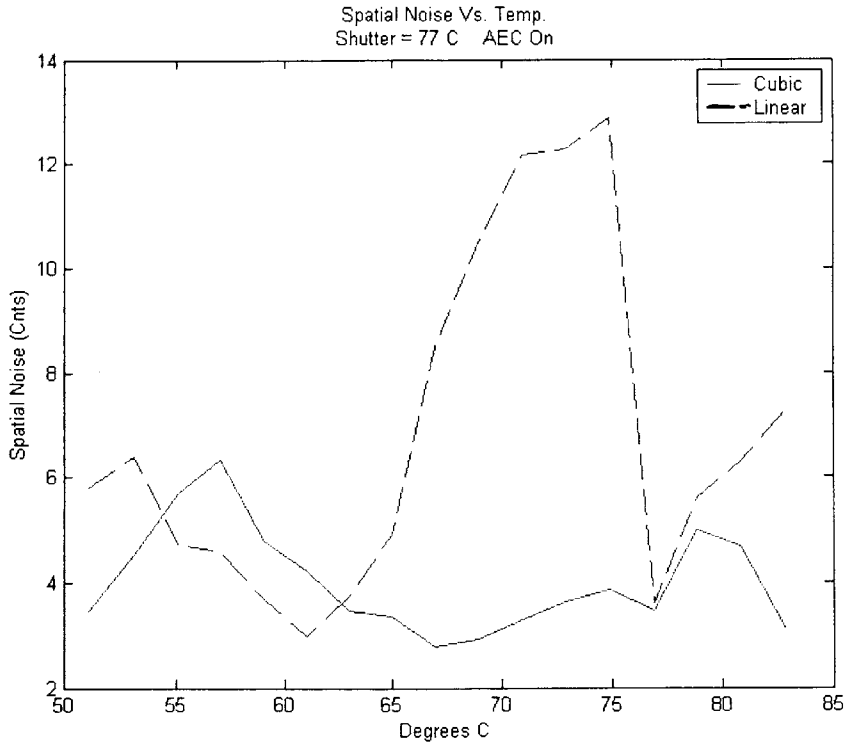


Figure 69: Spatial Noise Vs. Temp. for linear and cubic adjustments with AEC. Shutter = 77C

Overview

The graphs in Figures 70 and 71 below show much of the previous data in a more condensed form. The average of the spatial noise results for the three shutter temperatures in each range is computed. This was done for the AEC data as well. Both the no-AEC and AEC averages are plotted in each range for the linear (Figure 70) and cubic (Figure 71) adjustments.

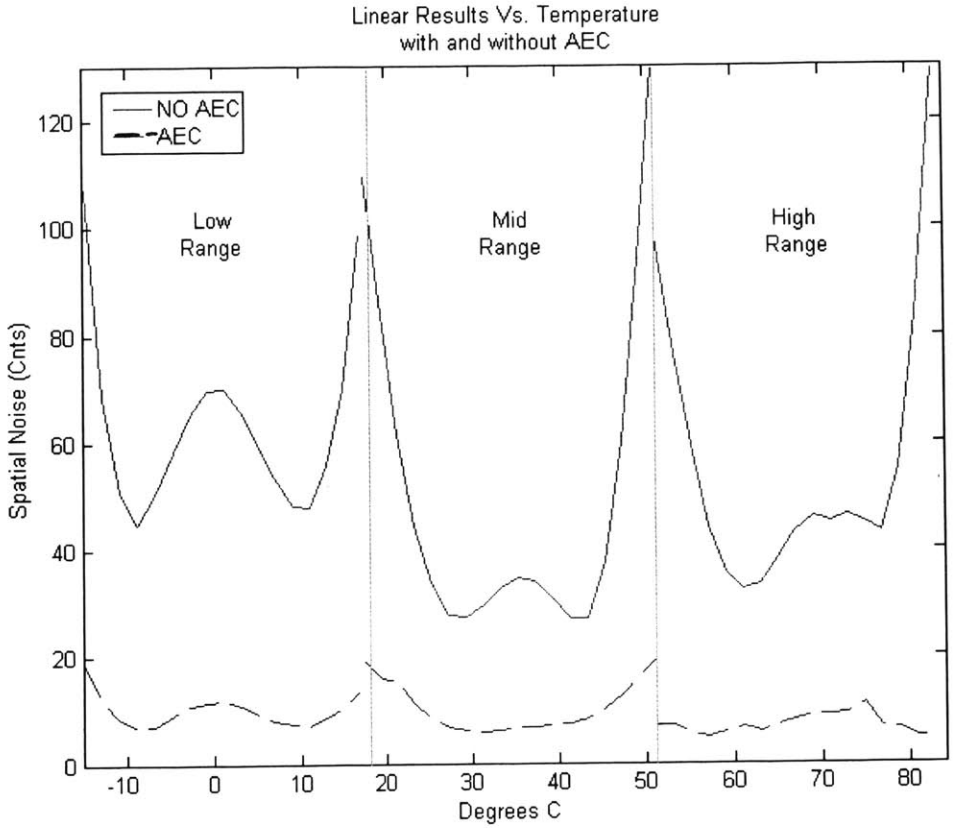


Figure 70: Average of Linear adjustment for each temperature range. Solid line is without AEC, Dashed is with AEC

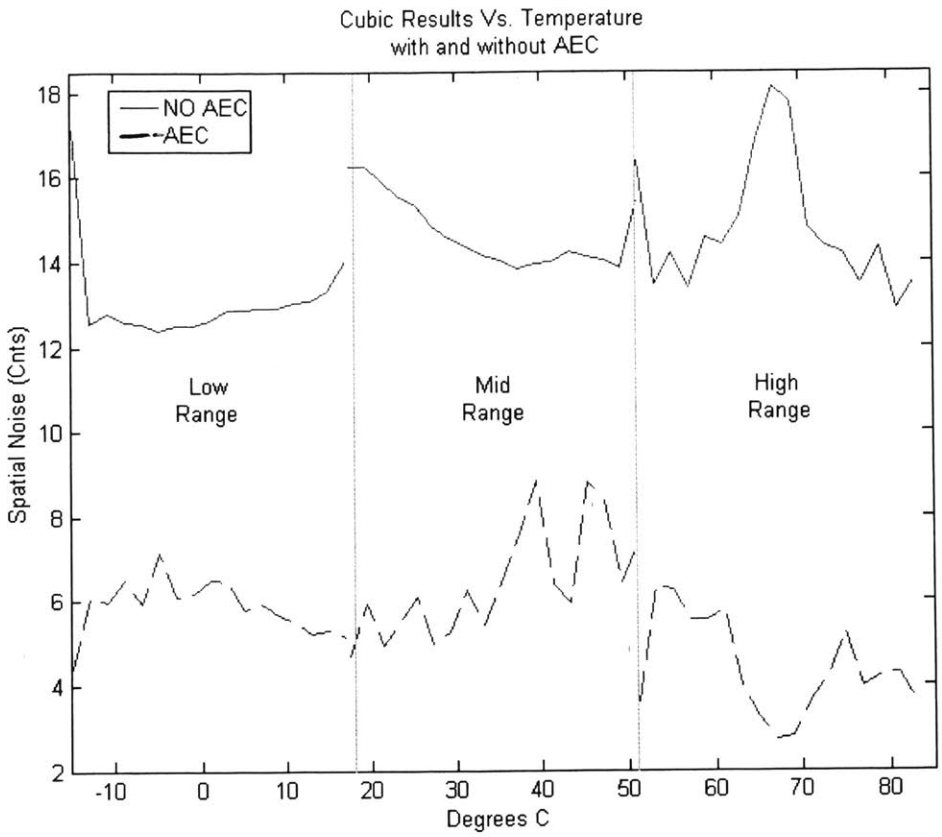


Figure 71: Average of Cubic adjustment for each temperature range. Solid line is without AEC, Dashed is with AEC

Further Investigation / Implementation

The results above show that the fine map update results in the spatial noise being lowered almost to the levels of normal TEC operation. There are several steps that should be taken if the exploration of TEC-Less operation is continued. First and foremost data from several more FPAs is needed. Second, an investigation into gain correction across temperature and ROIC temperature based gain needs to be done. Third, the algorithms shown in this report that were implemented on the PC need to be implemented in the firmware of the unit in order to drastically reduce test time and therefore allow improved appraisal of their effectiveness.

The data in this report is from one FPA only, which makes the results uncertain. More tests need to be done in order to validate the results and make them statistically

sound. The SIM220 module is designed to hold a CASPER FPA. The only FPA available at the time of these tests was a LAM2D FPA, which has a larger package and footprint than a CASPER, therefore making the integration of the LAM and the 220 FFE much harder. This combined with restrictions on time led to only one FPA being tested.

The *GO* correction and fine map adjustment algorithms can not be completely implemented on current hardware. However, the existing electronics may be able to implement a version of the algorithms that would allow for testing and analysis. The hardware would need several revisions, particularly in memory control and FPGA design.

In order to implement the *GO* adjustment algorithm the DSP would need to compute a fairly precise frame average every few frames and adjust the *GO* pots according to that average. The average is computed from raw FPA data so the average would need to be computed in DSP1, one of the signal processors on the VSP that has access to the raw data. That DSP currently computes a frame average from a video histogram but this average is very coarse and is not as precise as needed for this algorithm. DSP1 is also responsible for the other non-uniformity corrections (NUC) and has minimal available clock cycles for computing this algorithm.

Therefore implementation of the *GO* adjustment on a SIM220 module would require increased usage of the FPGA in one of two ways. First the frame average could be computed in the FPGA by implementing a large adder. If the frame average were computed by the FPGA, changing the pot settings would not be difficult. The second option is to implement fine correction in the FPGA therefore freeing up cycles in DSP1 that could be used to implement the *GO* adjustment algorithm. Implementing the fine correction in the FPGA is discussed in more detail in the next section.

Still other possible, though less desirable, ways to implement *GO* adjustment include freezing the image every 10 frames or so to allow the DSP time to compute the average of that frame and implement the algorithm. Obviously this would introduce an unwanted image artifact, but for testing purposes may be acceptable. Another option is to collect and store in the DSP1 DRAM a subset of pixels across the frame, and then compute the average of these pixels when clock cycles are available.

A last option might be implementing the algorithms on a newer SCC500 module. The SCC500 setup is quite different from SIM220 and already completes the NUC in the

FPGA as well as calculates a 33-bit frame sum. Calculating an average from this sum would not be hard, and could be used to implement the algorithm in the FPGA. The drawback of the SCC500 approach is that the auto-heating compensation and analog *GO* circuits are not present on the SCC500 FFE board as they are on the SIM220. Obviously if the digital approach works the analog *GO* adjustment is of no concern but the auto-HC circuit is crucial.

Implementing the Fine Map Adjustment

The fine map adjustment can most likely be implemented in the current 220 module for one sub-range of temperatures. The amount of memory required to store all the maps and coefficients needed to cover the entire operating range is much more than is currently available in the module. Figures 72 and 73 below show roughly the amount of storage needed for the standard operation and the linear and cubic fit adjustments. The amount of memory shown for the linear approach is under the assumption that three sub-ranges will cover the entire operating range as is the case with the cubic fit. This assumption is extremely unrealistic for the linear fit, as one slope matrix is only good for 5-7 degrees. It is represented in this way though for comparison purposes. It would be most beneficial if the cubic fit algorithm could be implemented, even if only for one temperature sub-range.

Coarse: 8-bit values: 320 x 240 x 8 = 614,400 bits = 76.8 Kbytes

Fine: 16-bit values: 320 x 240 x 16 = 1,228,800 bits = 153.6 Kbytes

Gain: 12-bit values: 240 x 320 x 12 = 921,600 bits = 115.2 Kbytes

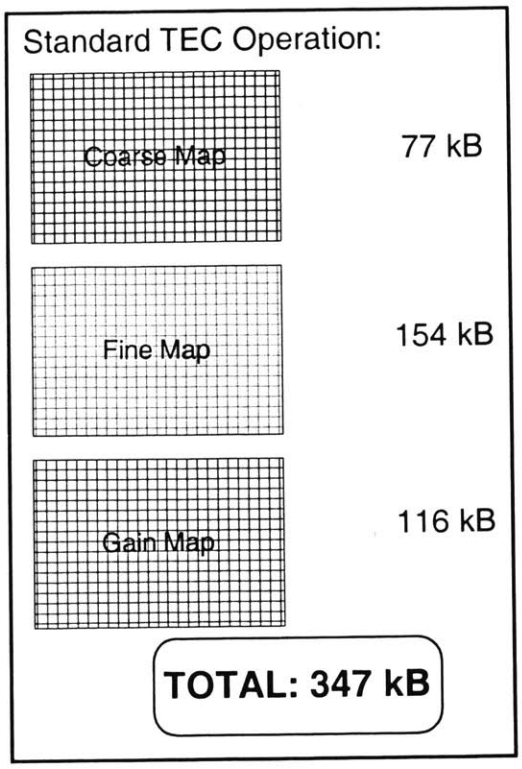


Figure 72: Memory usage for standard operation

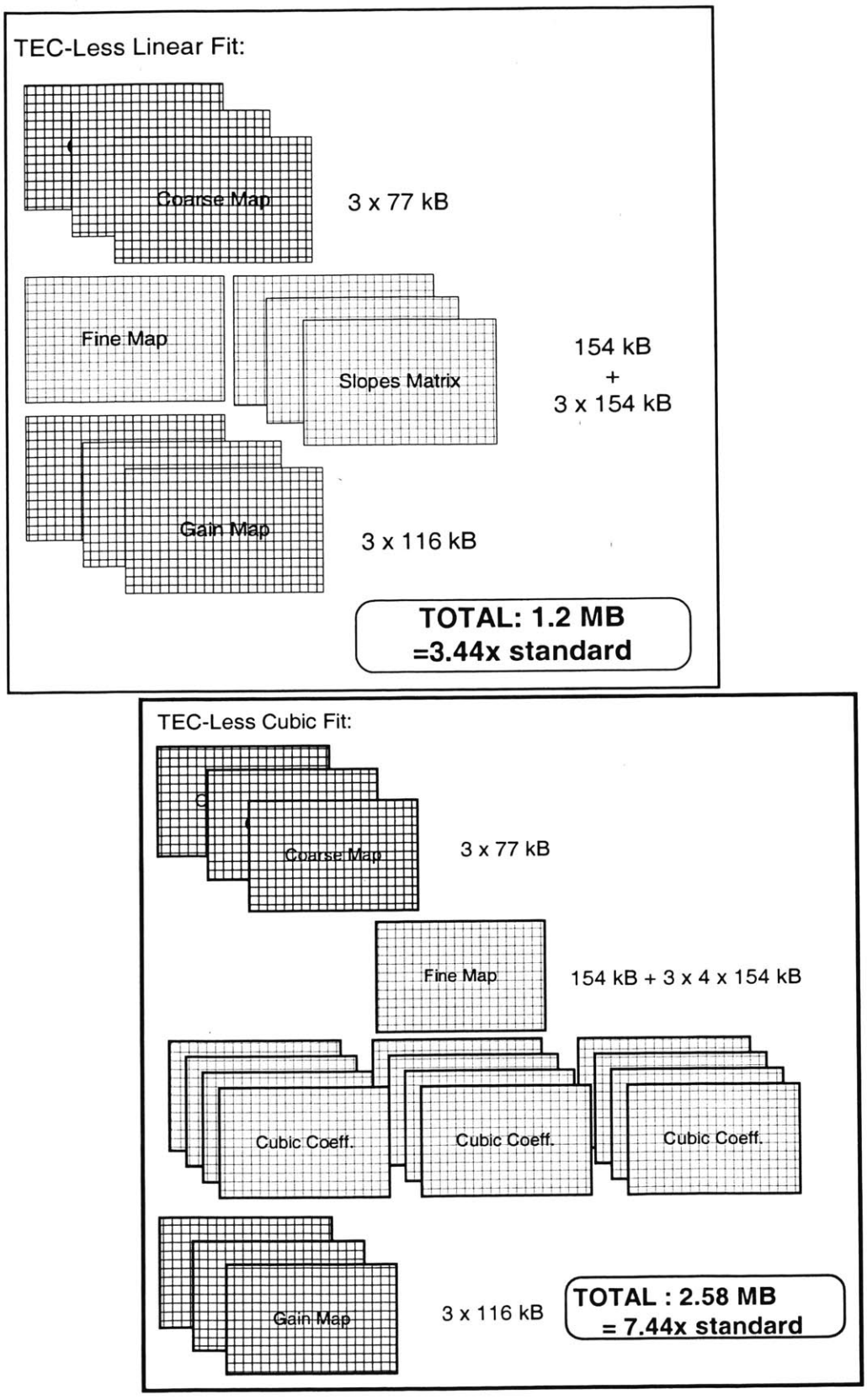


Figure 73: Memory usage for linear and cubic adjustments

In order to determine how the algorithm could be implemented, the current memory setup will be reviewed. There are two DSP chips and an FPGA on the VSP. These three chips are responsible for all video processing, but only DSP1 and the FPGA are responsible for pixel-by-pixel or NUC corrections. Both the FPGA and DSP1 are responsible for many other operations such as FPA timing, overlays, and TEC control. The current setup divides the pixel corrections between these two chips.

The FPGA is responsible for coarse offset correction. The FPGA has access to an external 1Meg x 16 bit DRAM. Within this DRAM the memory is organized into 4 pages. Each page has 64K addresses and 16-bit values. Pages 0 through 4 contain the coarse values, a test pattern, overlay1 and overlay2 respectively. The 8-bit coarse values are packed 2 to an address. Obviously these four pages take up far less memory than is available in the 1Meg RAM.

DSP1, on the other hand, is responsible for the fine and gain correction. DSP1 has access to a 1Meg x 24-bit DRAM. In this RAM the DSP stores program values, defect codes, the gain map and the fine map. Again there is a large portion of available memory but the real issue with the DSP is processing cycles. Even if all the maps and coefficients could be stored, DSP1 does not have enough open clock cycles after pixel substitution to allow calculation of new fine maps.

The solution, then, is to calculate fine as well as coarse in the FPGA. Later versions of FPGA programs, found in SIM210 and SCC500 modules, have the ability to do fine offset calculations. They are not, however, programmed to do map updates such as what is proposed here. Therefore implementation would need substantial FPGA work and memory control. Redoing the FPGA code and memory control would cost a sizeable amount of engineering hours and sufficient design work.

The fine map would need to be updated for small changes in ambient temperature, but assuming that the ambient is not changing very rapidly and the unit is somewhat insulated leads to the conclusion that the updates could be done every few frames and still be sufficient. The FPGA would most likely have time to update the map over several frames. Using a buffer in memory, the FPGA could calculate the new fine map piecewise putting a few newly calculated values in the buffer each frame and then switching the buffer to be the active map when finished.

The FPGA DRAM would be almost entirely full if the FPGA was used to do both the coarse and fine corrections. Figure 75 shows the page layout, with each page representing a 64K block. The 8-bit coarse maps fit in one block because they are packed 2 values to each address. The 16-bit fine map and coefficients, however, cannot fit in one page and are shown as taking two, even though they do not entirely fill both pages.

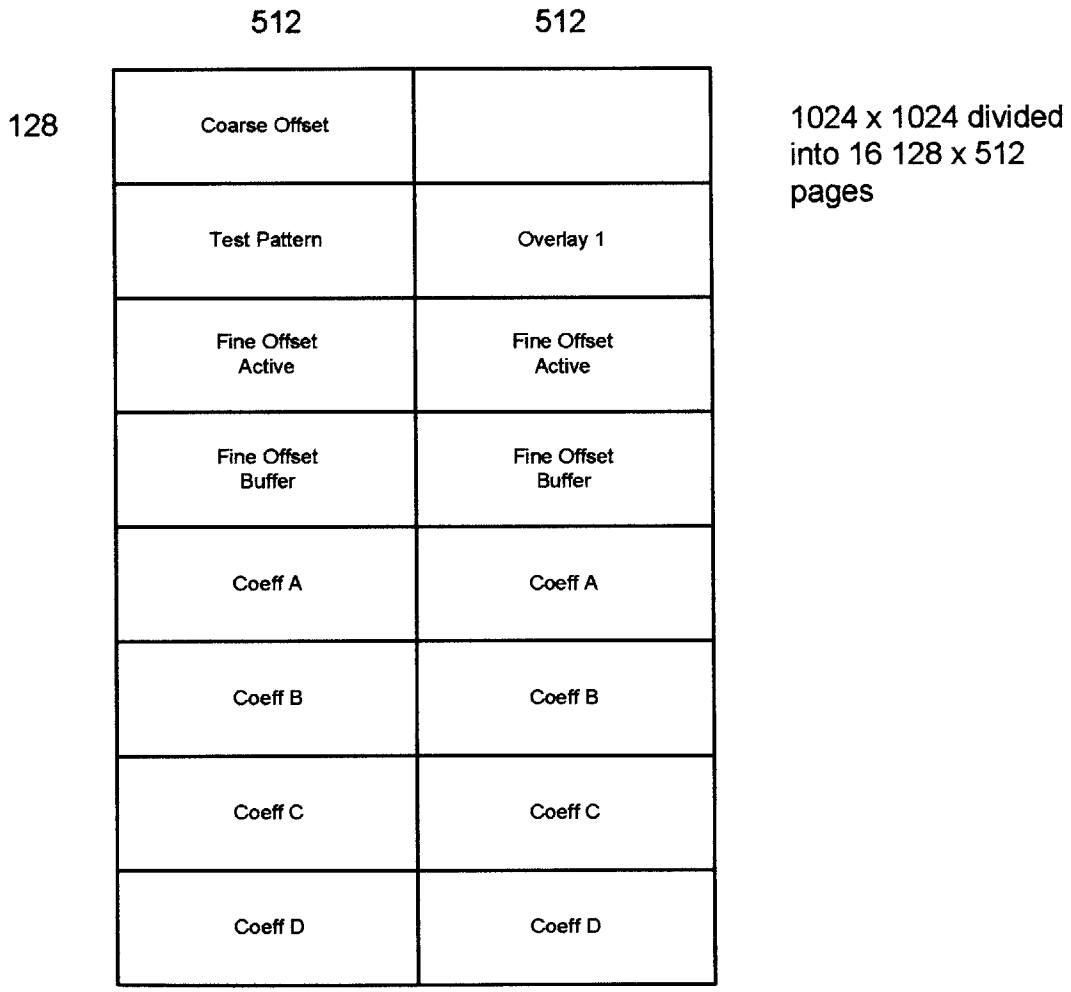


Figure 74: Pages of 1Meg x 16-bit memory for testing Cubic algorithm

The remaining hurdle to this test setup is the update of the constant coefficient at shutter. During shutter a normal fine map could be computed as normal by DSP1 and stored. The shutter operation could then be extended slightly to allow the DSP to compare the shutter fine map and the most recent active fine map. The constant coefficient could then be updated with the difference of the two.

In summary, the implementation of the fine map update for one temperature range depends on several variables. First, the FPGA must be reprogrammed and have the available resources after doing its normal computations to compute the cubic equation. With a lot of reprogramming this is most likely possible. Second, the FPGA DRAM must store all the coefficients, a fine map buffer, the active fine map, the coarse map, a test pattern and an overlay. The 1Meg DRAM would provide sufficient memory for this. Lastly the update of the constant coefficient at shutter must be completed. By extending the shutter time the DSP could most likely do these calculations.

Conclusion

A quick comparison between TEC and TEC-LESS spatial noise results shows that the TEC-LESS approach discussed in this report produces results close to TEC operation. The table below shows the average and best spatial noise test results for 581 TEC units at room temperature. The second column of the table lists the result of the TEC-LESS unit at room temperature. Notice that the result is comparable to the TEC average.

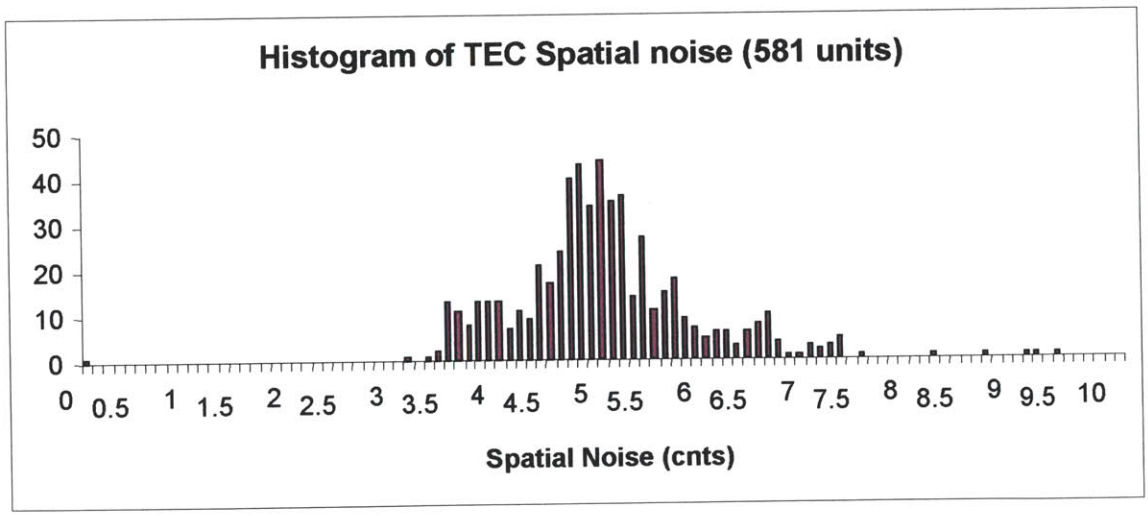


Figure 75: Histogram of TEC Spatial Noise results

TEC:		TEC-LESS
Avg. S.Cnts:	Best S.Cnts	Spatial Cnts
4.89	3.1	5.9

Table 5: Comparison of TEC and TEC-LESS spatial noise at room temp

Several initial steps were taken in the investigation of TEC-Less operation during this research. First, the digital algorithm for the update of *Global Offset* was shown to be more advantageous than the analog approach and is essential for keeping the pixels in range across temperature changes. Second, the constant power connection between bias and *GO* provided symmetric bolometer dynamics when changing temperatures. Finally the update of the fine offset map was explored in detail.

The results of the fine map adjustment show that a linear update would be sufficient over the small range of a few degrees, and if enough slope matrices could be stored, could be a viable option over larger ranges. However, the cubic adjustment worked well over 30 degrees and more by keeping the spatial noise fairly constant over the entire range. Implementation of the cubic adjustment algorithm would be most beneficial.

However, more data must be collected before anything is implemented in hardware. Although the results of the *GO* and fine map adjustments are promising, they are based on data from only one FPA and module. More FPAs and tests should be done to validate the results found here. If the cubic update continues to provide the same results, then a limited implementation using current hardware may be possible and was discussed in detail above. To implement the cubic adjustment over the entire operating range of temperatures would require a significant increase in available memory and FPGA and DSP changes.

Taking all of this into account, the conclusion that TEC-Less operation is possible is reasonable. But just to validate the information found here and implement a version in hardware will require a sizeable amount of engineering hours and redesign. In addition, other tests focusing on gain and possibly other unforeseen tests will need to be conducted as well. If implementation is successful, the calibration time for a TEC-Less unit will still most likely be several times longer than standard units. The trade-off between the savings of removing the TEC and this additional calibration time would need to be explored, as well as the proposed end use of the TEC-Less unit.

List of Abbreviations

AEC	Automatic Error Correction (global corrections)
BB	Black Body. Produces a uniform thermal field
CCA	Circuit Card Assembly
CMOS	Complimentary Metal Oxide semiconductor
DAC	Digital to Analog Converter
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor. Two DSPs present on SIM220 VSP
FFE	Focal-plane Front End CCA
FPA	Focal Plane Array
FPGA	Field Programmable Gate Array. One found on VSP CCA
GO	Global Offset voltage
LAM	Large Area
NTSC	National Television Standards Committee (typical North American standard)
NUC	Non-Uniform Correction (pixel-by-pixel corrections)
PAL	Phase Alternating Line (typical European standard)
PC	Personal Computer
ROIC	Read-Out Integrated Circuit CMOS
SAM	Small Area
SCC	Standard Camera Core
SIM	Standard Imaging Module
TEC	Thermal Electric Cooler
TEC-Less	a module running without a TEC, and therefore not thermally stabilized
TECSP	TEC Set Point. controls the potentiometer that sets the TEC temperature
UUT	Unit Under Test
VSP	Video Signal Processor CCA

References

Backer, Brian., Dr. Neal Butler, Dr. Margaret Kohin et al. "Recent Improvements and Developments in Uncooled Systems at BAE SYSTEMS North America," SPIE AeroSense Conference, Paper 4721-11. April 2002.

Cullen, Jay. "Electronics Hardware Video Signal Processor (VSP) Bottom-Up Design (BUD)" BAE SYSTEMS document. 1999.

Fonstad, Clifton G. Microelectronic Devices and Circuits. McGraw-Hill:New York, 1994. p.131-313.

Gray, Paul R. and Robert G. Meyer. Analysis and Design of Analog Integrated Circuits. John Wiley and Sons: New York, 1993. p.410-580.

Strang, Gilbert. Introduction to Linear Algebra. Wellesley-Cambridge Press:Wellesley MA, 1993. p.180-189.

Acknowledgements

Figure 1: http://www.ipac.caltech.edu/Outreach/Edu/infrared.html
Figure 2, 3, 4 and 6: Brian Backer, BAE SYSTEMS North America
Figure 74: Jay Cullen, BAE SYSTEMS North America

Special Thanks to:
BAE: Brian Backer
Neal Butler
Bob Smith
Scott Miller
Jason Timpe
Tom Polite
MIT: Prof. Qing Hu

Appendix A: Select C++ Code

A.1 GO Adjustment Algorithm

A.1.1 Reading TempOut

```
double CCollectFramesDlg::ReadTempOut(){
      static CGpib gpib("DEV14");
      gpib.Write("MEAS:VOLT?");
      gpib.Read(m_tempOut);
      return m_tempOut;
      }
```

A.1.2 Adjustment algorithm

```
const double coarseTargetVideo = 14336;
void CCollectFramesDlg::OnCalculateGlobalOffset(){
      CMyOfstream ofs;
ofs.MyOpen("c:\\Mahowald\\pots.xls", ios::out | ios::app | ios::ate);
      if (ofs.tellp() == (streampos)0) {
            ofs << "Tec\t";
            ofs << "Target\t";
            ofs << "scene avg\t";
            ofs << "Coarse\t";
            ofs << "Fine\t";
            ofs << "xFine\n";
      }
      vector<CFrame> frames(1);
      valarray<double> avgFrame(frame::NUM_PIXELS);
      UutCommands().Write("2019 2");
      UutCommands().Write("2005 4");
      try {
            UutCommands().Write("2300 3 00", false);
            int TecSP = UutCommands().ReadHexNumber(__FILE__, _
                                          _LINE__);
            UutCommands().ReadTillPrompt(__FILE__, __LINE__);
            UutCommands().Write("2300 2 00", false);
            int BiasCnts = UutCommands().ReadHexNumber (__FILE__,
            __LINE__);
            UutCommands().ReadTillPrompt(__FILE__, __LINE__);

                  for (int i = 1; i <= 10; i++){
                  FrameGrabber().CollectFrames(frames.begin(),
                        frames.end());
                  double scene, stdev;
                  GetAvgStdev(frames[0].m_data, scene, stdev);

                  UutCommands().Write("2300 1 00", false);
                  double coarseStart =
                  UutCommands().ReadHexNumber(__FILE__, __LINE__);
                  UutCommands().ReadTillPrompt(__FILE__, __LINE__);

                  UutCommands().Write("2300 1 80", false);
```

```
double fineStart =
      UutCommands().ReadHexNumber(__FILE__,
      __LINE__);
      UutCommands().ReadTillPrompt(__FILE__,
      __LINE__);

      UutCommands().Write("2300 3 80", false);
double xFineStart =
      UutCommands().ReadHexNumber(__FILE__,
      __LINE__);
      UutCommands().ReadTillPrompt(__FILE__,
      __LINE__);

m_tempOut = ReadTempOut();

double midpt = 128;
// FOR LOW TEMPS:
//double xFineStep = ((-.5529 * m_tempOut) + 2.7292);
//double fineStep = ((39.097 * m_tempOut) - 72.676);

//FOR HIGH TEMPS:
double xFineStep = ((-.5529 * m_tempOut) + 2.7292);
double fineStep = ((222.29 * m_tempOut) - 1196.0157);
double coarseStep = ((30001 * m_tempOut) -160793);

double xFine = xFineStart;
double fine = fineStart;
double coarse = coarseStart;

double diff = scene - coarseTargetVideo;
if (abs(diff) <= 50) {
      break;
      }

xFine = xFineStart + Round(diff/-xFineStep);
if ((xFine<0) || (xFine>255)){
      xFine = midpt;
      double predSceneA = (scene + (xFine-
      xFineStart)*xFineStep);
      diff = predSceneA - coarseTargetVideo;
      fine = fineStart - Round(diff/fineStep);
      double predScene = predSceneA + (Round(diff/-
      fineStep)*fineStep);
            if ((fine<0) || (fine>255)) {
                  fine = midpt;
                  predScene = (predSceneA + ((fine-
                  fineStart)*fineStep));
                  diff = predScene -
                  coarseTargetVideo;
                  coarse = coarseStart -
                  Round(diff/coarseStep);
                  predScene = predScene +
                  (Round(diff/-
                  coarseStep)*coarseStep);
            }
      fineStart=fine;
      diff = predScene - coarseTargetVideo;
```

```
                        xFine = xFineStart + Round(diff/-xFineStep);
                              if ((xFine<0) || (xFine>255)) {
                                    xFine = midpt;
                                    fine = fineStart -
                                    Round(diff/fineStep);
                                    predScene = predScene +
                                    (Round(diff/-fineStep)*fineStep);
                                    diff = predScene -
                                    coarseTargetVideo;
                                    xFine = xFineStart + Round(diff/-
                                    xFineStep);
                                    if (xFine<0){
                                          xFine=0;
                                    }
                                    else if (xFine > 255){
                                                xFine = 255;
                                          }
                                    }
                              }

                        UutCommands().Write(0x3006, 0, (int)coarse);
                        UutCommands().Write(0x3006, 1, (int)fine);
                        UutCommands().Write(0x3006, 5, (int)xFine);
                        UutCommands().Write(0x3007, 1, (int)coarse,
                        (int)fine, (int)BiasCnts, 1, (int)TecSP, (int)xFine,
                        false);
                        UutCommands().Write("8039", true);

                        ofs << TecSP << '\t';
                        ofs << coarseTargetVideo << '\t';
                        ofs << scene << '\t';
                        ofs << coarse << '\t';
                        ofs << fine << '\t';
                        ofs << xFine << endl;
                  }
                  ofs.MyClose();
            }
            catch(exception& e) {
                        AfxMessageBox(e.what());
            }
}
```

A.2 Fine Map Updates

A.2.1 Calculating Slope Matrix

```
      void CCollectFramesDlg::OnCalSlopes(){
      try {
            UutCommands().Write("2019 2");
            UutCommands().Write("2005 4");

            const int startTemp = 165; //TECSP counts
            const int stopTemp = 185;  // TECSP counts
            const int stepTemp = 5;
            const int numTemps = (stopTemp - startTemp)/stepTemp + 1;
            vector<CFineOffset> frames(numTemps);
```

```
        valarray<double> slope(frame::NUM_PIXELS);
        valarray<double> x(numTemps);
        valarray<double> y(numTemps);
        int i = 0;
        for (int t = startTemp; t <= stopTemp; t += stepTemp, ++i){
              UutCommands().Write(0x3006, 4, t);
              Sleep(5000);
              OnCalculateGlobalOffset();
              SceneShutter();
              UutMemory().ReadFine(frames[i].m_data);
              x[i] = m_tempOut;
              };
        for (int p = 0; p < frame::NUM_PIXELS; ++p) {
              for (int t = 0; t < numTemps; ++t) {
                    y[t] = frames[t].m_data[p];
              }
              double b;
              LeastSquaresFit(&x[0], &y[0], numTemps, slope[p], b);
              //slope[p] = (slope[p]/128);
        }

        CMyOfstream ofs;
        ofs.WriteColumnOrdered("c:\\Mahowald\\slopes", slope,
              frame::NUM_ROWS, frame::NUM_COLS);
        ofs.MyClose();

  }
  catch(exception& e) {
        AfxMessageBox(e.what());
        }
  }
```

A.2.2 Linear Update

```
  void CCollectFramesDlg::OnAutoAdjust(){
  try {
        UutCommands().Write("2019 2");
        UutCommands().Write("2005 4");

        OnCalculateGlobalOffset();

        valarray<double> slope(frame::NUM_PIXELS);
        CMyIfstream ifs;
        ifs.MyOpen("c:\\Mahowald\\slopes.col", ios::in |
        ios::binary);
        ifs.ReadColumnOrdered(slope, frame::NUM_ROWS,
        frame::NUM_COLS);
        ifs.MyClose();
        CMyOfstream ofs;
        ostringstream oss;

        static valarray<DWORD> fineOffset(frame::NUM_PIXELS);
        valarray<DWORD> newFineOffset(frame::NUM_PIXELS);

        if (m_shutter) {
              UutMemory().ReadFine(fineOffset);
              m_shutter = false;
```

```
                  oss.str("");
                  oss << "C:\\Mahowald\\testdata\\SpatialNoiseVerify
                        \\TestFrames";
                  oss << "\\FineMap";
                  ofs.WriteColumnOrdered(oss.str(), fineOffset,
                              frame::NUM_ROWS, frame::NUM_COLS);
            }
            m_tempOut = ReadTempOut();
            for (int p = 0; p < frame::NUM_PIXELS; ++p) {
                  newFineOffset[p] = fineOffset[p] + (m_tempOut -
                  m_shutterTemp) * slope[p];
            }
            UutMemory().WriteFine(newFineOffset);

            oss.str("");
            oss <<
"C:\\Mahowald\\testdata\\SpatialNoiseVerify\\TestFrames";
            oss << "\\PredFineMap";
            ofs.WriteColumnOrdered(oss.str(), newFineOffset,
                              frame::NUM_ROWS, frame::NUM_COLS);
            UutCommands().Write("2019 0");
            UutCommands().Write("2005 1");
      }
      catch(exception& e) {
                  AfxMessageBox(e.what());
      }

}
```

A.2.3 Calculating Coefficient Matrices

```
void CCollectFramesDlg::OnCreateCubicCoeff(){
      try {
            //UutCommands().Write("2019 2");
            //UutCommands().Write("2005 4");

            const int startTemp = 187; //TECSP Counts
            const int stopTemp = 255;
            const int stepTemp = 4;
            const int numTemps = (stopTemp - startTemp)/stepTemp + 1;
            vector<CFineOffset> frames(numTemps);
            valarray<double> slope(frame::NUM_PIXELS);
            valarray<double> CoeffA(frame::NUM_PIXELS);
            valarray<double> CoeffB(frame::NUM_PIXELS);
            valarray<double> CoeffC(frame::NUM_PIXELS);
            valarray<double> CoeffD(frame::NUM_PIXELS);
            valarray<double> xlin(numTemps);
            valarray<double> ylin(numTemps);
            Matrix x(numTemps,1);
            Matrix y(numTemps,1);
            Matrix sigma(numTemps,1);
            Matrix afit(4,1);
            sigma.set(.000001);
            int i = 0;
            int j = 1;
            CMyOfstream ofs;
            ostringstream oss;
```

```
            CMyIfstream ifs;

            // FOR COLLECTING FINE MAPS WITH TEC SP VARIATION
            oss.str("");
            oss << "C:\\Mahowald\\testdata\\CubicAdjust\\HighRange\\
                  FineForCoeff4";
            string directory = oss.str();
            _mkdir(directory.c_str());

            for (int t = startTemp; t <= stopTemp; t += stepTemp,
                  i++,j++) {
                  UutCommands().Write(0x3006, 4, t);
                  Sleep(5000);
                  OnCalculateGlobalOffset();
                  SceneShutter();
                  UutMemory().ReadFine(frames[i].m_data);
                  x(j) = ReadTempOut();
                  xlin[i] = x(j);
                  oss.str("");
                  oss << directory;
                  oss << "\\fine" << t;
                  ofs.WriteColumnOrdered(oss.str(), frames[i].m_data,
                  frame::NUM_ROWS, frame::NUM_COLS);
                  ofs.MyClose();
                  };

//Creat Cubic Coeff
            for (p = 0; p < frame::NUM_PIXELS; ++p) {
                  for (t = 0; t < numTemps; t++) {
                        y((t+1),1) = frames[t].m_data[p];
                  }
                  afit = CubicFit(x, y, sigma, 4, numTemps);

                  CoeffA[p] = (double) afit.get(4);
                  CoeffB[p] = (double) afit.get(3);
                  CoeffC[p] = (double) afit.get(2);
                  CoeffD[p] = (double) afit.get(1);
            }

            ofs.WriteColumnOrdered("c:\\Mahowald\\CubicCofA", CoeffA,
                  frame::NUM_ROWS, frame::NUM_COLS);
            ofs.MyClose();
            ofs.WriteColumnOrdered("c:\\Mahowald\\CubicCofB", CoeffB,
                  frame::NUM_ROWS, frame::NUM_COLS);
            ofs.MyClose();
            ofs.WriteColumnOrdered("c:\\Mahowald\\CubicCofC", CoeffC,
                  frame::NUM_ROWS, frame::NUM_COLS);
            ofs.MyClose();
            ofs.WriteColumnOrdered("c:\\Mahowald\\CubicCofD", CoeffD,
                  frame::NUM_ROWS, frame::NUM_COLS);
            ofs.MyClose();

      }
      catch(exception& e) {
            AfxMessageBox(e.what());
      }
}
```

A.2.3.1 Least Squares Cubic Fit

```
Matrix CCollectFramesDlg::CubicFit(Matrix x, Matrix y, Matrix sigma,
      int M, int lengthx){

      // Function to fit a polynomial to data :
      //   x        Independent variable
      //   y        Dependent variable
      //   sigma    Estimate error in y
      //   M        Number of parameters used to fit data

      // Outputs
      //   afit    Fit parameters; a(1) is intercept, a(2) is slope
      //   siga    Estimated error in the parameters a()
      //   yy       Curve fit to the data
      //   chisqr  Chi squared statistic

            //* Form the vector b and design matrix A

        int i, j, k, N = lengthx;
        Matrix b(N);
        Matrix A(N,M);
        for( i=1; i<=N; i++ ) {
            b(i) = y(i)/sigma(i);
            for( j=1; j<=M; j++ )
                  A(i,j) = pow(x(i),(double)(j-1))/sigma(i);
            }
        // Compute the correlation matrix C
        Matrix C(M,M);
        Matrix Cinv(M,M);
        for( i=1; i<=M; i++ ) {   // (C inverse) = (A transpose)* A
            for( j=1; j<=M; j++ ) {
              Cinv(i,j) = 0.0;
              for( k=1; k<=N; k++ )
                  Cinv(i,j) += A(k,i)*A(k,j);
                   }
            }
        C = inv(Cinv);  // C = ( (C inverse) inverse)

       //* Compute the least squares polynomial coefficients afit
      Matrix afit(M);
        for( k=1; k<=M; k++ ) {
            afit(k) = 0.0;
            for( j=1; j<=M; j++ )
             for( i=1; i<=N; i++ )
               afit(k) += C(k,j) * A(i,j) * b(i);
        }
      return(afit);
      }

Matrix CCollectFramesDlg::inv(Matrix A)

            // Compute inverse of matrix
```

```
// Input
//    A    -    Matrix A (N by N)
// Outputs
//   Ainv  -    Inverse of matrix A (N by N)
//  determ -    Determinant of matrix A   (return value)
{
  int N = A.nRow();
  assert( N == A.nCol() );
  Matrix Ainv(N,N);
  Ainv = A;  // Copy matrix to ensure Ainv is same size

  int i, j, k;
  Matrix scale(N), b(N,N);// Scale factor and work array
  int *index;  index = new int [N+1];
  //* Matrix b is initialized to the identity matrix
  b.set(0.0);
  for( i=1; i<=N; i++ )
      b(i,i) = 1.0;
      //* Set scale factor, scale(i) = max( |a(i,j)| ), for
      //each row
  for( i=1; i<=N; i++ ) {
      index[i] = i;   // Initialize row index list
      double scalemax = 0.;
      for( j=1; j<=N; j++ )
      scalemax = (scalemax > fabs(A(i,j))) ? scalemax :
      fabs(A(i,j));
      scale(i) = scalemax;
  }

 //* Loop over rows k = 1, ..., (N-1)
  int signDet = 1;
  for( k=1; k<=N-1; k++ ) {
      //* Select pivot row from max( |a(j,k)/s(j)| )
      double ratiomax = 0.0;
      int jPivot = k;
      for( i=k; i<=N; i++ ) {
        double ratio = fabs(A(index[i],k))/scale(index[i]);
        if( ratio > ratiomax ) {
            jPivot=i;
            ratiomax = ratio;
        }
      }

//* Perform pivoting using row index list
int indexJ = index[k];
if( jPivot != k ) {                // Pivot
  indexJ = index[jPivot];
  index[jPivot] = index[k];    // Swap index jPivot and k
  index[k] = indexJ;
  signDet *= -1;                // Flip sign of determinant
}
//* Perform forward elimination
for( i=k+1; i<=N; i++ ) {
  double coeff = A(index[i],k)/A(indexJ,k);
  for( j=k+1; j<=N; j++ )
      A(index[i],j) -= coeff*A(indexJ,j);
  A(index[i],k) = coeff;
```

```
            for( j=1; j<=N; j++ )
                b(index[i],j) -= A(index[i],k)*b(indexJ,j);
        }
    }
    //* Compute determinant as product of diagonal elements
        double determ = signDet;       // Sign of determinant
        for( i=1; i<=N; i++ )
        determ *= A(index[i],i);
    //* Perform backsubstitution
        for( k=1; k<=N; k++ ) {
            Ainv(N,k) = b(index[N],k)/A(index[N],N);
            for( i=N-1; i>=1; i--) {
              double sum = b(index[i],k);
              for( j=i+1; j<=N; j++ )
                  sum -= A(index[i],j)*Ainv(j,k);
              Ainv(i,k) = sum/A(index[i],i);
            }
              }
        delete [] index;      // Release allocated memory
            return(Ainv);
        // return( determ );
}
```

A.2.4 Cubic Update

```
void CCollectFramesDlg::AutoAdjustCubic(valarray<DWORD>& newFineOffset)
{
      OnCalculateGlobalOffset();

      valarray<double> CoeffA(frame::NUM_PIXELS);
      valarray<double> CoeffB(frame::NUM_PIXELS);
      valarray<double> CoeffC(frame::NUM_PIXELS);
      valarray<double> CoeffD(frame::NUM_PIXELS);

      CMyIfstream ifs;
      ifs.MyOpen("c:\\Mahowald\\CubicCofA.col", ios::in | ios::binary);
      ifs.ReadColumnOrdered(CoeffA, frame::NUM_ROWS, frame::NUM_COLS);
      ifs.MyClose();

      ifs.MyOpen("c:\\Mahowald\\CubicCofB.col", ios::in | ios::binary);
      ifs.ReadColumnOrdered(CoeffB, frame::NUM_ROWS, frame::NUM_COLS);
      ifs.MyClose();

      ifs.MyOpen("c:\\Mahowald\\CubicCofC.col", ios::in | ios::binary);
      ifs.ReadColumnOrdered(CoeffC, frame::NUM_ROWS, frame::NUM_COLS);
      ifs.MyClose();

      ifs.MyOpen("c:\\Mahowald\\CubicCofD.col", ios::in | ios::binary);
      ifs.ReadColumnOrdered(CoeffD, frame::NUM_ROWS, frame::NUM_COLS);
      ifs.MyClose();

      CMyOfstream ofs;
      ostringstream oss;

      static valarray<DWORD> fineOffset(frame::NUM_PIXELS);
```

```
if (m_shutter) {
      UutMemory().ReadFine(fineOffset);
      for (int p = 0; p < frame::NUM_PIXELS; ++p) {
      CoeffD[p] = CoeffD[p] + (fineOffset[p] -
      ((pow(m_shutterTemp,3)*CoeffA[p])+(pow(m_shutterTemp,2)*
      CoeffB[p])+(m_shutterTemp*CoeffC[p])+CoeffD[p]));
      m_shutter = false;
      }
}
      m_tempOut = ReadTempOut();
      for (int p = 0; p < frame::NUM_PIXELS; ++p) {
      newFineOffset[p] =
      (pow(m_tempOut,3)*CoeffA[p])+(pow(m_tempOut,2)*CoeffB[p])+(
      m_tempOut*CoeffC[p])+CoeffD[p];
      }

UutMemory().WriteFine(newFineOffset);
}
```

A.3 NOTES

Other C++ code was used to change the TEC temperature, run the thermal chamber and automate tests. That code has not been included in this report.

Appendix B: Select Matlab Code

B.1 Spatial Noise Calculation

```
function output = spatial(twoDarray)

% +-------------------------------------------------------------------+
% + SPATIAL Takes a frame fed in by the variable 'array' and          +
% +         calculates the spatial noise                              +
% +                                                                   +
% +         Jason T. Timpe - Lockheed Martin IR Imaging Systems       +
% +         Brian Backer   - Lockheed Martin IR Imaging Systems       +
% +                                                                   +
% +         Design Date: 6/5/98                                       +
% +                                                                   +
% +         Revisions                                                 +
% +         1.  Added functions for input variable and auto           +
% +             of the frame size - Backer (7/14/98)                  +
% +         2.  Revised code for the calculation of the temporal of   +
% +             noise - Backer (7/14/98)                              +
% +-------------------------------------------------------------------+

% get size of input frame
rows = size(twoDarray,1) ;
columns = size(twoDarray,2) ;

% INITIALIZE VARIABLES
std_dev = 0 ;
i=0; j=0 ;
counter = 0 ;
a = 0 ;
b=zeros(2852,1);
z=0;

% GET STD OF EACH 5X5 BLOCK IN FRAME AND STORE IN ARRAY a
for i=1:(rows/5)-2                % row index
   for j=1:(columns/5)-2          %column index
      z=z+1;
      counter=counter+1 ;
      x=(5*i)+1 ;
      y=(5*j)+1 ;
      subarray = twoDarray(x:x+4,y:y+4) ;   % get subframe and stats
      b(z,1) = std2(subarray);
      a = a + b(z,1) ; % a = running total of spatial counts
      end
   end

% CALCULATE THE MEDIAN SPATIAL NOISE OF THE FRAME
% output = a / counter ; %Average Spatial Noise
 output = median(b);
%output = b;
```

B.2 Create Slopes Matrix

```
function slopes = CreateSlopeMatrix(FineMaps,Tempout)
size =length(Tempout);
for i = 1 : 240
    for k = 1 : 320
        [P,S] = polyfit(Tempout,reshape(FineMaps(i,k,:),1,size),1);
        slopes(i,k) = P(1);
    end
end
```

B.3 Linear Adjustment Predictor

function newfine = AutoAdjust(FineMaps,slopes,shutterTemp,offset,TempOut)

```
% given actual finemaps collected across temperature and
% a slopes matrix, function will produce a new fine map predicted
% from linear fit and compare to original fine maps
% Mahowald

numTemps = size(FineMaps,3);
for i = 1 : numTemps
    newfine(:,:,i) = offset + ((TempOut(i) - shutterTemp).*slopes);
end

Rowfactor = 50;
Colfactor = 50;
cycles = 2;
Figure;
for i = 1 : cycles
    for k = 1 : cycles
        for t = 1 : numTemps
        Actual(1,t) = FineMaps(Rowfactor*i,Colfactor*k,t);
        Linear(1,t) = newfine(Rowfactor*i,Colfactor*k,t);
        hold on;
        end
        plot(TempOut,Actual);
        plot(TempOut,Linear,'--r');
        title('Actual Fine Map Values and Linear Prediction');

    end
end
```

B.4 Least Squares Cubic Fit

```
function coeff = CreateCubic(FineMaps,Tempout)

% Returns a 240x320x4 matrix with cubic coefficients
% calculated for each pixel
% Mahowald 11/02

size =length(Tempout);
for i = 1 : 240
    for k = 1 : 320
        [P,S] = polyfit(Tempout,reshape(FineMaps(i,k,:),1,size),3);
        coeff(i,k,:) = P;
```

```
    end
end
```

B.5 Cubic Adjustment Predictor

```
function newfine =
CubicAdjust(FineMaps,CubicCoeff,TempOut,ShutterScene,ShutterTemp)
% given actual finemaps collected across temperature and
% a coeff. Matrix, function will produce a new fine map predicted
% from cubic fit and compare to original fine maps
% Mahowald

numTemps = size(TempOut,2);

for i = 1 : 240
    for k = 1 : 320
    CubicCoeff(i,k,4) = CubicCoeff(i,k,4) + (ShutterScene(i,k)-
polyval(CubicCoeff(i,k,:),ShutterTemp));
    newfine(i,k,:) = polyval(CubicCoeff(i,k,:),TempOut);

    end;
end;

Figure;
Rowfactor = 50;
Colfactor = 50;
cycles = 2;
Figure;
for i = 1 : cycles
    for k = 1 : cycles
        for t = 1 : numTemps
        Actual(1,t) = FineMaps(Rowfactor*i,Colfactor*k,t);
        Linear(1,t) = newfine(Rowfactor*i,Colfactor*k,t);
        hold on;
        end
        plot(TempOut,Actual);
        plot(TempOut,Linear,'--r');
        title('Actual Fine Map Values and Linear Prediction');

    end
end
```

B.6 TempOut to Degrees Conversion

```
function temp = TempOutToDeg(tempOut)
% approximate temperature conversion from Temp Out for
% LAM2D ROIC.
temp = -46.74*(tempOut) +348.23;
```

B.7 Notes

Other Matlab functions used to plot and compare data were designed but are not included in this report.