Characterization of Hyperspectral Imagers

Jonathon Michael Wade

HYPERSPECTRAL IMAGER CHARACTERIZATION

by

Jonathon Wade

A Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Major: Electrical & Computer Engineering

The University of Memphis
August 2023

Contents

List of Figures

ACKNOWLEDGMENTS

The authors would like to thank Thomas Watson, Ryan Williamson, and Dr. Alfredo Ramirez for their invaluable discussions and help with the Pika and making working calibration files. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-21-2-0294. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

ABSTRACT

Hyperspectral imaging (HSI) is a technology that captures three dimensions of information, two spatial dimensions and a spectral dimension. A focal plane, which is used to collect the intensity information of a scene, is only two-dimensional. In a conventional camera this is not a problem as it collects only two-dimensional images, but since hyperspectral imagers collect three-dimensional images this presents a problem. How do you collect these three-dimensional images? There have been clever schemes to use the focal plane to collect three-dimensional data as well as different capture methods which make either a spatial or spectral dimension the third dimension that is collected with subsequent scans. The next logical question is how to characterize and calibrate these systems. A radiance calibration is described and results in a low error calibration. For most systems, established methods will not work therefore, new metrics need to be developed and new merits created to fully measure and calibrate these systems. For line scan systems, a stationary measurement of a knife edge is not possible and movement is necessary for capturing the target. Two methods, stare step and continuous, were used and the results were similar to each other. In this paper, 3D noise methodology is considered for hyperspectral imagers (HSI), but there are not three dimensions to evaluate for an HSI as the spectral dimension cannot be used. Therefore, the concepts from 3D noise are used to discover that the spatial noise of the imager is the main source of noise in the sensor.

Keywords: Hyperspectral, Characterization, Calibration, MTF, Noise, Radiometric Calibration

1. INTRODUCTION

Hyperspectral Imaging (HSI), also referred to as imaging spectrometry, is an exciting area of activity at the moment as various industries are finding many uses for HSI. Hyperspectral imaging was initially used for remote sensing applications with NASA's Airborne Imaging Spectrometer (AIS) and airborne visible-infrared imaging spectrometer (AVISIR)[1] which was mainly used for identifying minerals on the Earth's surface. Since then numerous different disciplines have taken advantage of this technology to further research in their fields. For example, in agriculture, HSI has been used to determine the health of crops leading to higher yields and quality of produce.[2] This has allowed for farming to be done on a plant-to-plant basis at a large scale since the imagery can pinpoint plants with diseases or other conditions detrimental to the crops. Other applications include but are not limited to medical imaging,[3] environmental monitoring,[4] remote sensing for mineral identification,[5] and food quality and safety.[6]

Hyperspectral Imaging devices are sensors that are able to capture accurate and high resolution spectral data. A general definition of a hyperspectral imager is an imager that is capable of imaging more than 30 different contiguous spectral bands. Each band includes a certain range of wavelengths, and the end of one band is the start of the next in hyperspectral imagers. The shorter the band interval the higher the spectral resolution. For a given resolution, the number of bands determines the bandwidth of the HSI. The final image is a 3D image with two spatial dimensions and one spectral dimension. A focal plane array (FPA) is what captures the images in a conventional camera. FPAs are normally made up of many detectors arranged in a rectangle. The FPA is a 2D object as the word plane in the name would suggest. This presents a problem, as a conventional camera does create a 3D data cube, but one of the dimensions is time. With HSI we would like to capture three dimensions at the same time on a 2D focal plane. There have been numerous solutions to this problem that are presented in section 2.

While hyperspectral sensors are used in many different industries taking images would be of no use if the sensor is not correctly calibrated and characterized. For example, if one wished to detect the spectrum of a mineral but did not have an accurate wavelength calibration for the sensor it would be impossible to correctly identify and detect the mineral. For this reason, accurate calibration and characterization is needed to obtain useful data from the sensor. Without a calibrated sensor data becomes less useful and cannot be tied to physical units which is described in section 3.

The characteristics are another important aspect to each individual sensor. Even though two sensors can be the same model the performance can vary greatly between the two. Hyperspectral sensors are working on the scale of nanometers or micrometers and must be built precisely to achieve good performance, but because of the small errors made each system will be unique and requires that every system produced be examined for deficiencies. If parts of the system are not aligned properly this can lead to undesired shifts or magnifications of the light received at the focal plane of the detector. An example of this would be measuring the spectral distortion, also known as ”smile”, which is a shift of the wavelength information on the focal plane leading to erroneous results.

The issue of how to characterize and calibrate these systems correctly is a topic of research. Previously others have given the theoretical ad mathematical background of the measurements as in Hyperspectral Remote Sensing by Michael Eismann.[7] There is not much information about how to carry these measurements out. Riley Logan from the University of Montana has previously characterized and calibrated a Pika-L for produce.[6] He gives a methodology and verification of his measurements, but it has some vagueness and decisions that are not fully justified. Another group led by Jablonski,[8] was seeking to outline theoretical background and methodologies to take these characterizations and calibrations. It is quite technical and easy to get lost in the paper if you are not very familiar with practices used in the field. A The IEEE P4001 standardization group is developing a general guide for characterizing HSI in the reflective bands. Some concepts and techniques carry over from traditional imagers, but care must be taken to ensure that these methods are not blindly applied. The largest difference between HSI and traditional imagers is that after taking sequential frames a traditional sensor captures a data cube with two spatial dimensions and a temporal dimension. The contribution I wish to make with this thesis is to outline the theory and methodologies behind my measurements with minimal arbitrary decisions. I would also like to investigate adapting known broadband methodologies of characterization and calibration to HSI. With traditional imagers you are able to see how a 2D scene changes over time. HSIs produce a data cube with two spatial dimensions and a spectral dimension. With an HSI you can see the spectral information of a 2D scene.

2. HYPERSPECTRAL IMAGER ARCHITECTURE

Snapshot Systems

A snapshot hyperspectral camera is one that is able to capture the three dimensions, the two spatial dimensions and one spectral dimension, at one time. This is the ideal situation for collecting images. As you collect more images a video can be made just as one is from a conventional sensor. This type of sensor is most like the modern electro-optical or infrared cameras used today, except that it has many more channels. There are many different architectures that have been developed to utilize a two dimensional focal plane to capture three dimensions of data.

The idea of a snapshot hyperspectral imager dates back to 1938 with the Bowen image slicer.[10] This

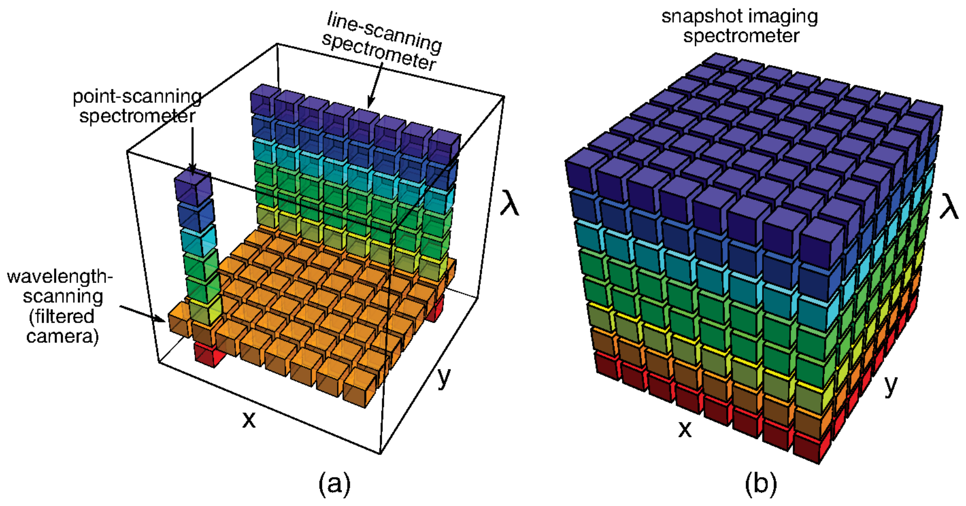
line-scanning spectrometer
point-scanning spectrometer
wavelength-scanning (filtered camera)
λ
x
y
(a)
snapshot imaging spectrometer
λ
x
y
(b)

Figure 1. Representation of the different architectures of HSI[9]

method used mirrors to make an image into a long strip to emulate the input slit. This not only allows three dimensions to be imaged at once, but also increases the amount of light that enters and the spectral width of the image. Since then researchers have been working on improving this technique[11] while other schemes have also been used to try to represent three dimensional data on a two dimensional focal plane. There are two different methods that have been used to achieve this.

One method groups the incoming light into separate images before the focal plane. Using this method multiple duplicate images are created on the focal plane in the different bands or wavelengths. This can be achieved in different ways. One method utilised optical fibers to split the image up into smaller images. An example of this system is called Densepak.[12] Another implementation uses a microlens array to split the images in a similar manner.[13] There have been improvements on this method as well.[14] Using a beam splitter, the RGB light field, which contains the angular information of a scene, can be collected while an encoded hyperspectral image is also taken at high spectral resolution. This is similar to using detectors on the FPA as bits of information to be decoded later. The images are then processed to create a high spatial and spectral resolution image from the two sets of data.

The second method uses clever schemes on the focal plane to place the spectral information for each pixel collected similar to image slicer. By rearranging all of the data on the focal plane the full 3D cube can be realized. One of the first methods, called computer tomography, takes an intensity image and several projections from different angles on the focal plane using a kinoform grating. From this data tomography can be used to reconstruct the original image with its spectral components.[15]

The limitation of this architecture at the moment is resolution. Since focal planes are only two dimensions either the spatial or the spectral resolution must be sacrificed to gain the ability to capture three dimensions of data in one scan. In the future, as focal planes become larger and better schemes are developed this will mitigate this issue and allow for this architecture to become more widely used in many fields, but for now it is used in few applications and has yet to be widely adopted.

Spectral Scanning Systems

A scanning system can only record two dimensions in one scan instead of three as a snapshot system can collect. A spectral scanning system captures one band at a time, and with subsequent scans it stacks different bands to create the same three dimensional image the snapshot system produces. The obvious downside is that it takes more time and scans to acquire the same data as the snapshot systems, but two key benefits are that the spectral scanning systems are generally easier to manufacture and cost less to produce than a snapshot systems.

In a spectral scanning system the focal plane must be exposed to only one band or wavelength at a time to capture a two dimensional spatial image that is one band. This has been achieved a couple ways with the most straightforward being to use band pass filters on a filter wheel. As the name implies, these filters only allow one band to pass through them and reject the rest. Tunable filters, such as liquid crystal tunable filters, have also been used and have the benefit of speed as there is no need for a filter wheel.[16] By having a mechanism to quickly switch the filter in the system one can take images with different filters and stack them to create a hyperspectral image.

The other method that has been used to achieve a single band is a Fabry-Perot (F-P) interferometer. An F-P is a device that contains two mirrors positioned so that the light that enters will reflect multiple times before exiting the interferometer. Only wavelengths that have constructive interference will be passed. By tuning the position of the mirrors, the peak wavelength that exits the F-P can be changed acting similarly to a monochromator. Usually an additional filter is needed to reject harmonics of the desired wavelength.[17]

The advantage of spectral scanning over snapshot is the higher resolution since the entire focal plane is used for imaging instead of storing spectral information. The obvious disadvantage, as stated earlier, is that it takes more time to collect the same data that the snapshot system would collect. In addition to this, the scene would need to be stationary so that the different bands are not shifted. If anything in the scene moves then the comparison between two bands will suffer as you are not measuring the object in the same location in both images.

Spatial Scanning Systems

Spatial spanning is similar to spectral scanning, but the difference is that it scans across one spatial dimension and collects all bands in one scan. The output of one scan is a spatial line of pixels or a single pixel with the full spectral content. Subsequent scans with a spatial scanner stack the spatial rows to create the full three dimensional data cube. Spatial scanners suffer from the same disadvantage as spectral scanners in that it takes more time than the snapshot system to collect the entire data cube. There are two main types of spatial scanning those being whisk broom and push broom.

The first type, whisk broom, captures all the spectral data for a single spatial pixel at a time. The first hyperspectral camera, the AVIRIS from NASA, was a whisk broom sensor. It uses a mirror that is able to tilt to scan the across axis or perpendicular to the path of the satellite. One advantage of this sensor is that there is only one detector that needs calibration and all samples use the same path to get to the detector.[18] The disadvantages of this sensor type are that the scanning takes time. Another disadvantage is that it has moving parts that can wear and break.

The second type, is a push broom style camera that scans one spatial dimension and one spectral dimension. This is an improvement over the whiskbroom because it only needs to scan one direction. Push broom scanners use both dimensions of the focal plane with one being the spectral dimension and the other being the spatial dimension. This architecture has been widely used on aircraft and satellites since the imager needs motion along one spatial dimension to create the full three dimensional data cube. Push broom systems use a slit to create the spatial line which then enters either a grating or other dispersive optical device to separate the wavelengths from each other to then be projected on the focal plane[19] as shown in figure 2.

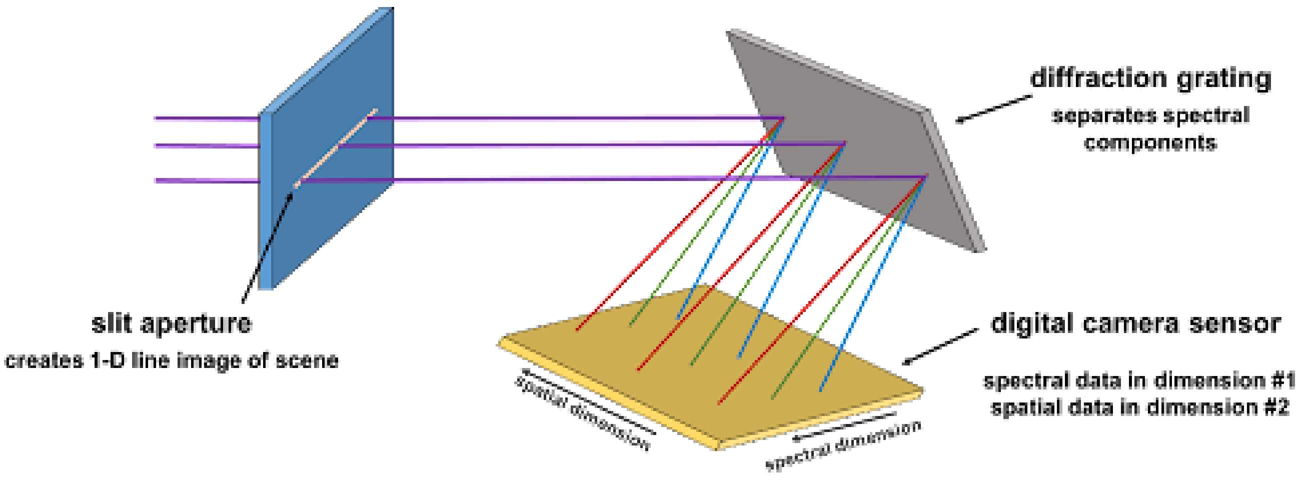
diffraction grating
separates spectral components
slit aperture
creates 1-D line image of scene
digital camera sensor
spectral data in dimension #1
spatial data in dimension #2
spatial dimension
spectral dimension

Figure 2. Working principle of a dispersive element HSI[19]

Spatial scanning has resolution advantages over snapshot imagers and push broom style sensors have the advantage of no moving parts. Motion is required for spatial scanning systems to acquire the full picture. This means that these systems are the perfect devices for remote sensing; whereas spectral scanning is more suited to static imaging such as medical imaging or laboratory imaging. Since the research group this work is for is working with remote sensing, we have chosen a push broom style camera for the research.

3. CHARACTERIZATION OF HYPERSPECTRAL IMAGERS

Characterization and calibration of hyperspectral sensors are necessary to ensure that good quality data is collected. Characterization can be thought of as investigation of the properties of a system such as looking at the amount of noise inherent in the system. Calibration is the process of converting the output of the system, usually digital numbers or grey counts, to physical units such as radiance or spectral radiance in a scene.

A wavelength calibration is the most important calibration that must be done on a hyperspectral imager. It is the mapping from detectors of the focal plane to wavelength. This calibration is achieved by presenting the sensor with a narrow band light, the narrow band light must be less than the spectral resolution of the camera, and the digital number peak is mapped to a pixel. This is repeated multiple times and a least squares linear or a quadratic fit is created to map pixel number to each wavelength.[8] A linear fit can be represented as:

$$y = mx + b\,, \tag{1}$$

y is the wavelength, m is the scaling factor, x is the pixel number, and b is the offset on the focal plane. A quadratic fit can be represented as:

$$y = ax^2 + bx + c \tag{2}$$

where y is the wavelength(or band), a is the quadratic coefficient, b is the linear coefficient or linear scaling component, c is the constant coefficient or shift, and x is the detector number on the focal plane. If a is zero then the fit becomes a linear fit rather than a quadratic. Ideally the wavelength calibration is a linear transform, but as can happen in the real world it may be slightly nonlinear as with the hyperspectral imager used later in this paper. A least squares fit is a curve fitting method that estimates the equation of a line by minimizing the difference between the sample points and the predicted value, also known as the residuals. If $\hat{y_i}$ is defined as:

$$\hat{y_i} = mx_i + b \tag{3}$$

Where $\hat{y_i}$ is the predicted y value from a linear fit. The least square regression can be expressed by the equation:

$$\arg\min_{m,b}(\sum_{i=1}^{n}(y_i - \hat{y_i})^2) \tag{4}$$

It finds the m and b that minimizes the summation of all residuals. This results in the best estimate of the relationship between two variables, in this case those are the detectors on the focal plane and the bands.

Radiometric calibration is used to make the conversion from digital number to spectral radiance. There are two types of radiance calibration, absolute and relative.[8] Absolute calibrations use National Institute of Standards and Technology (NIST) traceable equipment to ensure an accurate and ground truth calibration. This is needed to create data that has physical units and can be easily translated and compared to other systems. Relative calibrations will give data with no ground truth scale, but it will still yield the correct spectral curve shape for objects that are imaged.[8] Relative calibration may be good enough if all that is required is identifying spectral curves in a scene. Relative calibrations are also easier and less expensive to measure than absolute calibration meaning that time and money can be saved if absolute scale is not needed. Radiometric calibrations require that a known spectral source be recorded at different intensities to create a linear fit from digital number to spectral radiance. Generally radiance calibrations are also least square linear fits just like the wavelength calibration.

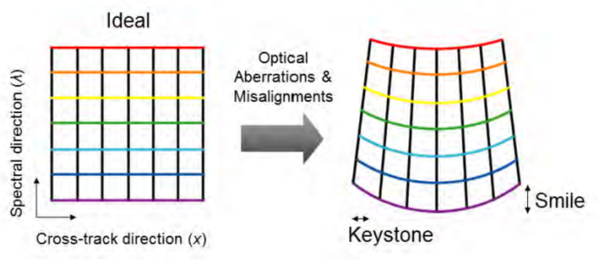
Ideal
Spectral direction (λ)
Cross-track direction (x)
Optical Aberrations & Misalignments
Smile
Keystone

Figure 3. Keystone and Smile on a Focal Plane[20]

Keystone and smile are two characterizations of hyperspectral systems that quantify the aberrations and quality of optical alignment in a system. Keystone is a measure of the spatial distortion on the focal plane.[9] It manifests as a trapezoidal shape where the top of an image projected on the focal plane is wider than at the bottom. Keystone is indicative of improper alignment or non-unity magnification in the system. Smile is a measure of the spectral distortion on the focal plane.[9] As seen in figure 3, the bands are curved across the focal plane resulting in a smile shape hence the name. Smile is the result of optical aberrations in the system[20] and is very serious since if the wavelength is incorrect it can lead to incorrect spectrum results and misclassification. Keystone and smile are quantified by the range of pixels covered by a field slit or a single wavelength respectively. This can be expressed as:

$$\text{Keystone} = (L_{\lambda\max} - L_{\lambda\min}) \tag{5}$$

$$\text{Smile} = (C_{\lambda\max} - C_{\lambda\min}) \tag{6}$$

Where L_λ is the center pixel of the slit target at a given wavelength and C_λ is the wavelength at a given test slit.[21] A test slit is a simple optical element used for various calibration and measurement purposes. It is a narrow, straight aperture in an opaque material. The primary function of a test slit is to create a well-defined and controlled light source or to modify the incoming light in a specific way. Equations 5 and 6 can be thought of as the range that a single wavelength or test slit covers. For example, when measuring smile a single wavelength is presented to the HSI and the center pixel is mapped across the focal plane for each spatial pixel. Ideally the center pixel does not move therefore $C_{\lambda\max}$ is equal to $C_{\lambda\min}$ and the smile is zero for the HSI. This is usually not the case and the center pixel will vary for different spatial pixels.

A group has been created by IEEE, labeled as P4001, to create standards for the characterization and calibration of hyperspectral imaging devices. Anyone can join the meetings and is headed by staff from Labsphere, the University of Oslo, and Clyde Hyperspectral Imaging and Technology. The group plans to cover performance characteristics, test methods, metadata, use cases, and terminology.[22] This group has meetings regularly presents updates every year at SPIE Defense and Commercial Sensing. I would like to mention that P4001 is moving towards using a sampling point spread function (SPSF) to test the spatial resolution of HSI instead of the Modulation Transfer Function (MTF). To capture the SPSF one would use a slit target that has a broadband source (or spectral source if you want to take the measurement for each band individually). The measurement technique has the user scanning the slit over at least seven different spatial channels (pixels). At each spatial channel the user needs to take at least 10 equally spaced sample points of the slit that are within the Full Width Half Max (FWHM) of the spatial resolution of the pixel and each of the 10 or more samples must have at least 100 frames for averaging. This data collection results in what is called the sampling line spread function (SLSF), which is similar to what is collected from a knife edge target. After subtracting dark frames from the data, one must extract the pixel intensity data for the different mirror positions. The 2D SPSF can be approximated by the orthogonal multiplication of the along track and across track SLSF,[9] which is represented in figure 4. I refer the reader to the P4001 document for complete details of target selection and measurement methodologies.[9] Later in this thesis I do use MTF as a way to measure the spatial resolution of the sensor, but this is because one of the goals of the paper was to try adapting broadband techniques to HSI.

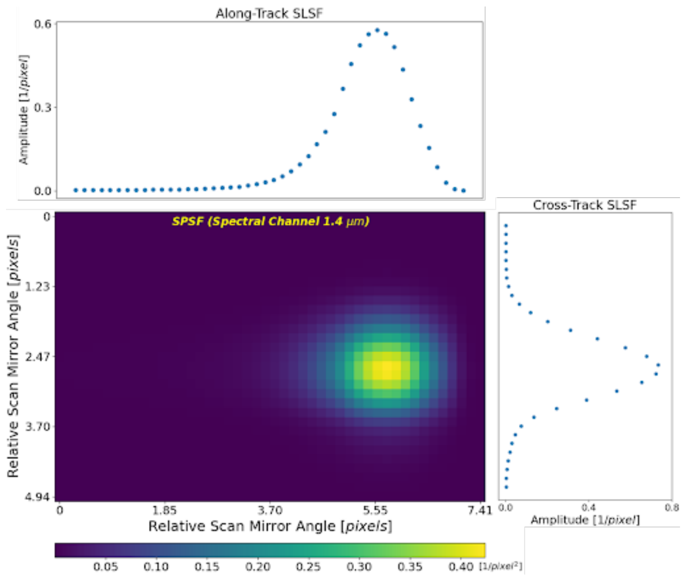

Figure 4. Orthogonal multiplication of SLSF to produce the SPSF[20]

4. CHARACTERIZATION OF THE PIKA-L

Pika-L

The camera used for the experiments was the Pika-L from Resonon which is a push broom or linescan style camera. The specifications are listed in Table 1.

Table 1. Pika-L Specifcations.

Spectral Range	387.9 - 1029.18 nm
Spatial Pixels	900
Spectral Pixels	300 (binned from 600)
Spectral Resolution FWHM	3.3 nm
Maximum frame rate	249 fps

The working principle of the Pika is light enters a slit and is diffracted by a diffraction grating onto a focal plane array, in this case it is an acA1920-155um Basler Ace CMOS camera. One dimension of the focal plane is for the spatial pixels and the other dimension of the focal plane is for the spectral pixels (or bands) as described in the spatial scanning section. Utilising the full extent of the two dimensions of the focal plane, it is able to capture a full row of spectral data across a horizontal spatial dimension in a single acquisition frame.

The Pika is delivered with a wavelength calibration from the manufacturer, meaning that the transform from focal plane detector to band has already been completed. Resonon uses a quadratic fit for this calibration. Ideally the wavelength calibration should be linear, but Resonon uses a quadratic fit to account for nonlinearities in the system. The quadratic coefficient is much smaller than the linear or constant coefficient indicating the calibration is very nearly linear.

Calibration Instrumentation

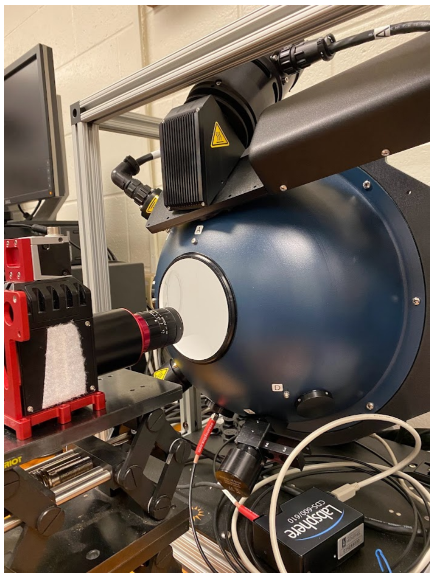

Figure 5. Integrating sphere with a Pika-L looking at the output port

Calibration of an electro-optical sensor requires a radiometric and spectral standard. An integrating sphere(see Figure 5), is a uniform lambertian surface meaning that the amount of light received by a detector is independent of the angle between it and the surface reflecting light. It also creates a flat, or uniform, field for the imager to view. This means that all of the spatial pixels are receiving approximately the same values. Manufacturers typically provide the non-uniformity report for each sphere produced (see Figure 6).

Labsphere also provided a measure of uniformity and non-uniformity deviation calculated as:

$$\text{Uniformity} = 1 - \frac{L_{max} - L_{min}}{2 * L_{mean}} * 100\,, \tag{7}$$

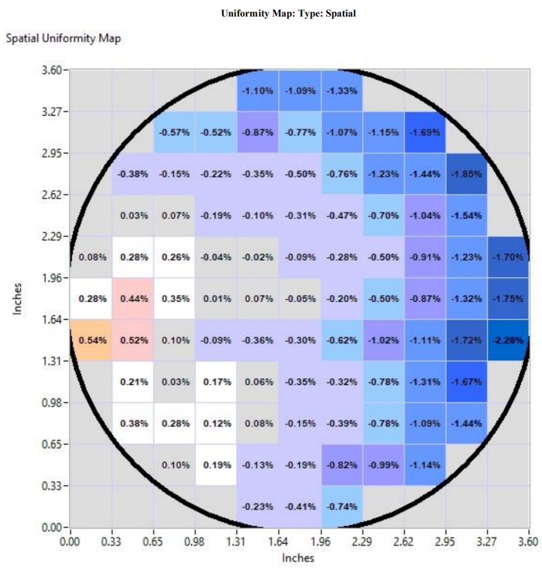

Figure 6. Integrating sphere uniformity[23]

$$\text{Non-Uniformity Deviation} = \frac{\text{StdDev}}{\text{Average}} * 100\,, \tag{8}$$

where L is luminance. Our sphere's uniformity is 98.59% and its non-uniformity deviation is 0.99%.[23]

The equipment used was an integrating sphere (USLR-D12F-NANS-P) with a broadband halogen bulb (HES-150) used as a light source and an attenuator to control the light level in the sphere. The bulb also had documentation detailing the spectral radiance curve of the bulb that is NIST traceable. This is important for determining the uncertainty in the calibration as it is dependent on the uncertainty of the bulb and sphere.

In the integrating sphere there is also a spectrometer that has been calibrated to measure the total radiance in the sphere. The calibration of the spectrometer was accomplished by taking the current read outs given by Labsphere at different attenuator settings. Taking the Labsphere data, which is in spectral radiance, and integrating it yields the total radiance in the sphere. After this is done, a linear fit can be made between the the current of the detector and the radiance in the sphere. The spectral distribution in the sphere remains constant but the magnitude of the spectral radiance at each wavelength will be scaled by the attenuator setting. This means one can have a known spectral curve for different light intensities by varying the amount of the bulb exposed to the sphere.

Radiometric Calibration

A radiometric calibration is the process of taking the output of the imager, digital numbers, and turning them into meaningful data, in this case radiance (and spectral radiance). This similar to a signal intensity

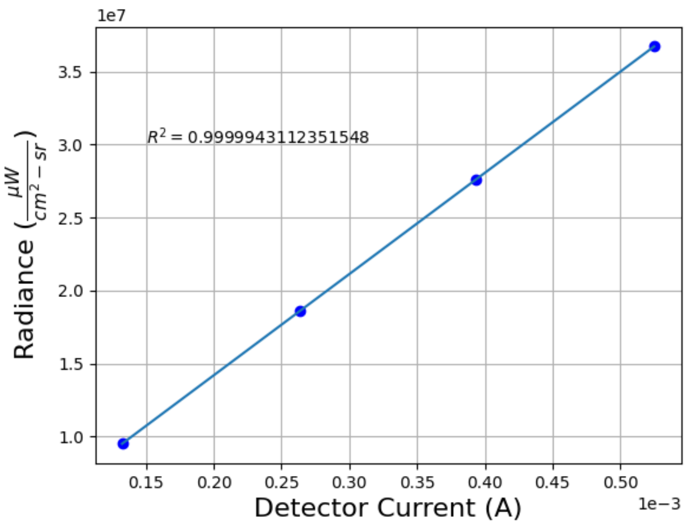

Figure 7. Linear fit for detector current to radiance

transfer function (SITF) used in thermal imaging cameras. Radiance is the spectral radiance integrated over the wavelengths of interest. Spectral radiance is typically measured in microflicks ($\frac{\mu W}{cm^2 - sr - \mu m}$), with radiance being in ($\frac{\mu W}{cm^2 - sr}$).

For the radiance calibration, the HSI was pointed at the four inch port of the sphere, and a total of 10 cubes were captured at different light levels using the attenuator as described in the previous section with care being taken not to capture saturated pixels while also keeping the signal from being too low in each band. Each cube had 100 scans of the flat field, 900 spatial pixels, and 300 spectral bands.

After the data was collected, the data processing can begin. The first step is to take a temporal average of the cubes. This helps to eliminate the temporal noise introduced by the HSI. For this research, 10 data cubes that have one scan, 900 spatial pixels, and 300 spectral bands were used for the radiance calibration. After interpolating the spectral curve given by Labsphere, the spectral radiance at any arbitrary wavelength in the sphere at any given radiance will be known. This is important because the Pika-L does not measure at integer wavelengths. Integration of the interpolated spectral radiance is needed to calculate the total radiance in the sphere at the time Labsphere made their measurements. It is important that the cutoff wavelengths match the exact spectral range of the imager to achieve accurate results. The interpolated spectral radiances are then scaled to the measured radiance of the Pika's data by dividing by the total radiance from Labsphere's data and then multiplied by the radiance observed from the calibrated spectrometer as shown in .

$$SSR = SR * (OR/LR) \tag{9}$$

Where SSR is the scaled spectral radiance, SR is the spectral radiance from the Labsphere provided data, OR is the observed radiance, and LR is the radiance from the provided Labsphere data. After making 270,000 linear fits (900 spatial pixels by 300 spectral bands), all of the coefficients needed to transform the digital number output of the camera to radiance or spectral radiance are obtained. The correction can be applied by multiplying the coefficient by the pixel intensity at the same position to get out the radiance data.

To verify the calibration was correct, it was applied to a video taken using a different light source in the integrating sphere (HIS-350).This is another bulb that came with the integrating sphere and is also NIST traceable. The same spectral radiance documents the manufacturer provided for the first bulb were also provided for this bulb. After the radiance calibration is applied to the raw data, a comparison can be made between the calibrated data and the manufacturer reported spectral radiance of the second bulb. Plotted below is the magnitude of the percent error between the corrected data (in spectral radiance) and the theoretical spectral radiance in the sphere.

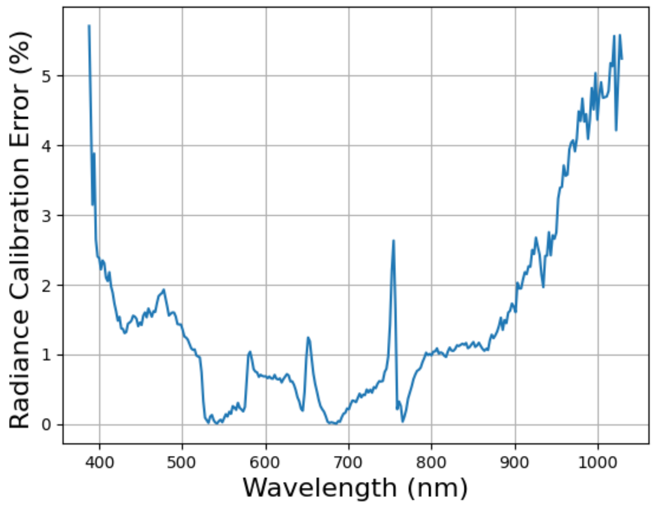

Figure 8. Magnitude percent error of radiance calibration

The minimum error was 0.006%, the average error was 1.48%, and the maximum error was 5.71%. From observing the graph, it can be seen that the error tends to be the highest at the high and low end of the spectrum. This could be influenced by the lower spectral response of the HSI at these wavelengths. The lower bands struggle even more because the bulb emits the least amount of light at those wavelengths.

Noise

The idea of 3D noise is used to analyze the noise in the system. 3D noise was developed by researchers from the U.S Army Night Vision laboratory in the early 1990's.[24] It is the idea that the total noise can be characterized in an imager by breaking down the different dimensions of a three dimensional data cube. To break down the dimensions of the images directional averaging is used. The three dimensions used for this are horizontal (image rows), vertical (image columns), and temporal (frames). These directional average operations are denoted as D_h, D_v, and D_t. If one of the directional averages are applied the three dimensional data is reduced to two dimensions. By applying two directional averages on three dimensional data it becomes one dimensional. For example, if a video is being examined and a temporal average (average all of the frames of the video) and then a horizontal average(average all rows of the temporal average image) what remains is a single row which is one spatial dimension. Assuming noise is independent and additive a flat field image can be modeled by:

$$U_{tvh} = S + N_t + N_v + N_h + N_{th} + N_{tv} + N_{vh} + N_{tvh} \tag{10}$$

Where U_{tvh} is the data in the data cube, S is the global average for the data cube, and N_t, N_v, N_h, N_{th}, N_{tv}, N_{vh}, N_{tvh} are the seven different zero mean noise components of the different dimensions of the noise cube. t refers to the temporal dimension, h refers to the horizontal (row) dimension, and v refers to the vertical (column) dimension. The N terms with multiple dimensions have noise components that are not captured in the single dimension. For example N_{tv} is the temporal column noise. It looks at how the noise in each column changes over time. We notate a directional average in the vertical spatial dimension as D_v, in the horizontal spatial dimension as D_h, and in the temporal dimension as D_t. Using this notation, the temporal noise of the data cube can be described as:

$$N_t = (1 - D_t)D_h D_v U \tag{11}$$

$$N_t = D_h D_v U - D_h D_v D_t U \tag{12}$$

S can also be expressed as:

$$S = D_h D_v D_t U \tag{13}$$

Therefore, using equation 13 we can rewrite equation 12 as:

$$N_t = D_h D_v U_{tvh} - S \tag{14}$$

The temporal noise is found by taking the two directional averages on the data cube and subtracting the global mean from it. This can be extrapolated to the vertical and horizontal dimensions to measure their noise values as well.

Notice that the noise is additive, but be careful on what that means. Going back to probability, if two variables, A and B, are independent then:

$$var(A) + var(B) = var(A + B) \tag{15}$$

In other words, noise variances are additive therefore the total noise can be calculated as (see reference[25]):

$$\sigma_{total} = \sqrt{\sigma_{tvh}^2 + \sigma_{tv}^2 + \sigma_{th}^2 + \sigma_{vh}^2 + \sigma_t^2 + \sigma_h^2 + \sigma_v^2}\,, \tag{16}$$

As seen above, knowing all seven components, temporal (σ_t), fixed row (σ_h), fixed column (σ_v), random spatial (σ_{vh}), temporal row (σ_{th}), temporal column (σ_{tv}), and random spatial-temporal (σ_{tvh}), of the noise will lead to the overall noise in a video or data cube. To isolate the noise components from each other, averages are taken across the dimensions that are not being investigated at that time. For example, if the desire was to calculate random spatial noise, all of the frames of a video would be averaged together and the standard deviation taken.

The data that needs to be captured for noise calculations is a uniform scene that is stable with time. The same data for the radiometric calibration can be used to calculate the noise in the system. Again using the data cube that is 100 scans of the flat field, 900 spatial pixels, and 300 spectral bands noise can be calculated in each band. In this case, there are only two dimensions to measure, spatial and temporal. This means that the 3D noise model is not required, but can be used to see where the noise is originating from in a sensor.

The Pika-L actually has 600 bands, but software binning is used to combine two bands into one by adding the two together. Since the Basler camera used is a 12-bit system the max digital number possible is 4095. After the software binning is done the maximum value is now 8190. The noise calculations are done after this binning. This is the noise that the camera outputs and is the relevant measure of the systems performance.

The overall noise in each band at all ten light intensities collected for the radiance calibration is quantified. These were found by isolating each band as a 2D spatial image and taking the standard deviation at each band meaning there are 90000 samples for each band as there are 100 scans and 900 spatial pixels. As the

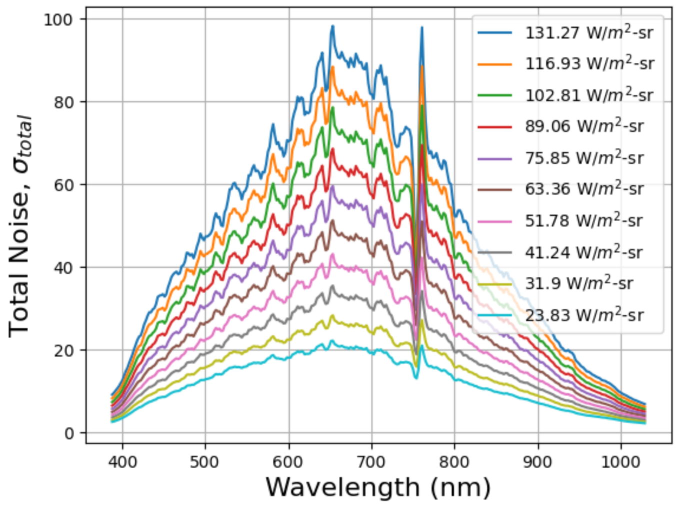

Figure 9. Random noise in the data cubes

figure 9 shows, the noise increases as light level increases which would indicate that there is signal dependent noise. There is also an anomaly at around 760 nanometers. Examining a second Pika-L unit, the peak at 760 nm was not present. This means it is likely a unique characteristic of the system and points to the need for calibration of individual systems.

The temporal component of the noise is found by averaging all 900 of the spatial pixels and taking the standard deviation from the resulting 100 samples as there were 100 temporal scans. The temporal noise trends up as the light level increases. The magnitude of the temporal noise is quite low compared to the total noise in the system as seen in Figure 10. The total noise was 29.26 counts on average between all the bands and data cubes. For the temporal noise, it was 0.64 counts on average. There is only the fixed column noise and the temporal column noise remaining.

Figure 11 shows the spatial, or pixel to pixel, noise in the system. Taking note of the magnitude and looking back at the total noise it can seen that a majority of the noise is coming from the non-uniformity in the pixels.

The temporal column noise can be calculated from the previous noises as:

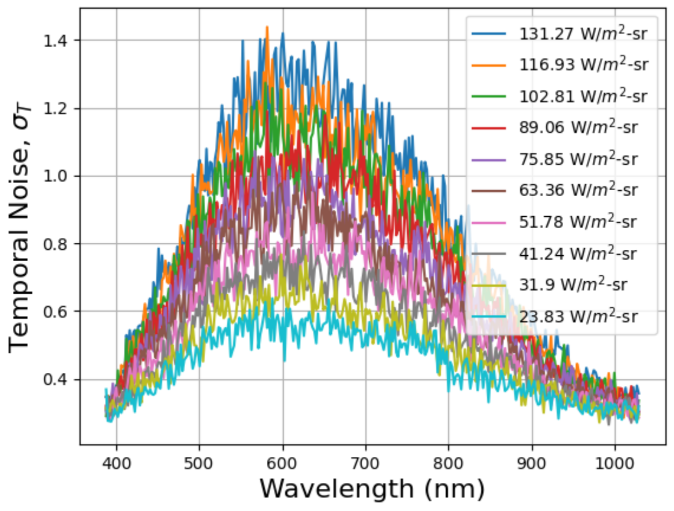

Figure 10. Temporal noise in the data cubes

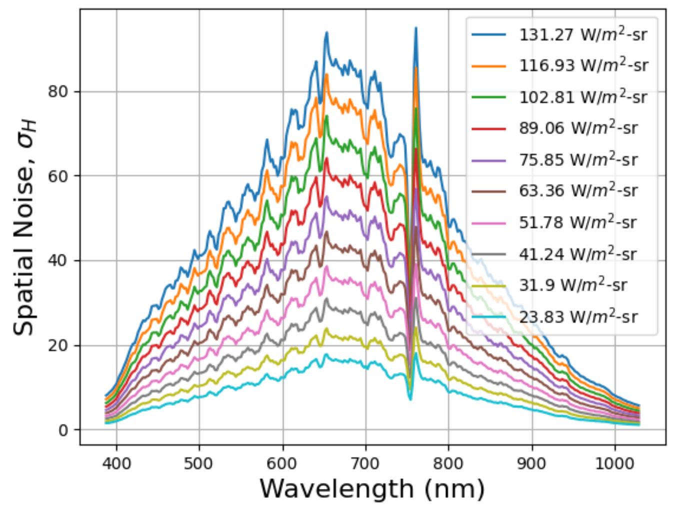

Figure 11. Spatial noise in the data cubes

$$\sigma_{TH} = \sqrt{\sigma_{Total}^2 - \sigma_T^2 - \sigma_H^2}\,, \text{[25]} \tag{17}$$

σ_{TH} can be thought of as the how much the column noise changes with time. It is the second highest of the noise sources with an average digital number value of 14.39 which is 24% of the total noise. The percent noise is calculated in equation 18

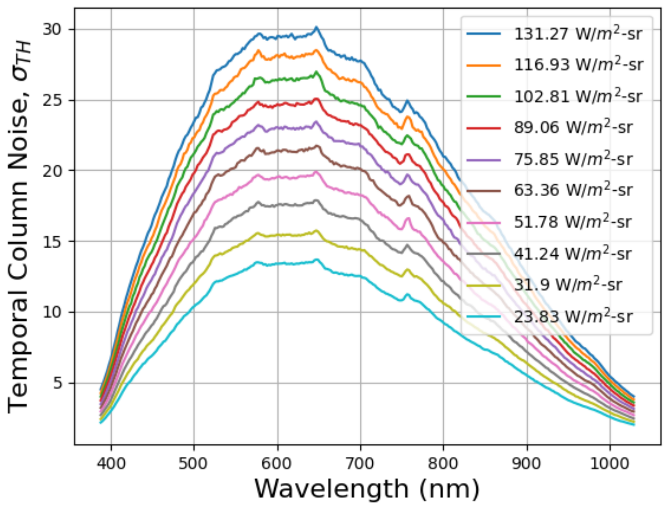

Figure 12. Temporal Column noise in the data cubes

$$\textbf{Percent of Total Noise} = \frac{\sigma_{TH}^2}{\sigma_{Total}^2}, \tag{18}$$

The average total noise was 29.26 counts and the average spatial noise was 25.13 counts. That means that roughly 74% of the total noise is from the pixel-to-pixel or spatial noise. This necessitates that a non-uniformity correction be applied to mitigate the noise in this dimension. For the non-uniformity correction (NUC), a two point NUC was used. The process for this is to take two videos, one at a dark setting and one at a bright setting, then make a linear fit for each spatial pixel, and in this case every band for each spatial pixel, to the same number. A global average was taken for each band and linear fits for each pixel and band to the average pixel value were made.

As can be seen in the scale of the vertical axis in Figure 13, the magnitude of the spatial noise was decreased by a factor of approximately 10. The notch at around 760 nm is still present but much less pronounced, and there is now a new anomaly at 700 nm, but the noise is still greatly reduced. There is still the dependence of the noise to light level. This may be a result of shot noise that is inherit when measuring photons, although this has not been tested or proven. Another technique that could have been utilized is to slightly rotate the HSI as it was taking the data cube to help remove any non-uniformity from the integrating sphere.[6] The spatial uniformity of the sphere seemed sufficient so this step was omitted, however it would be interesting to take data both with and without rotation to see the difference in the noise levels.

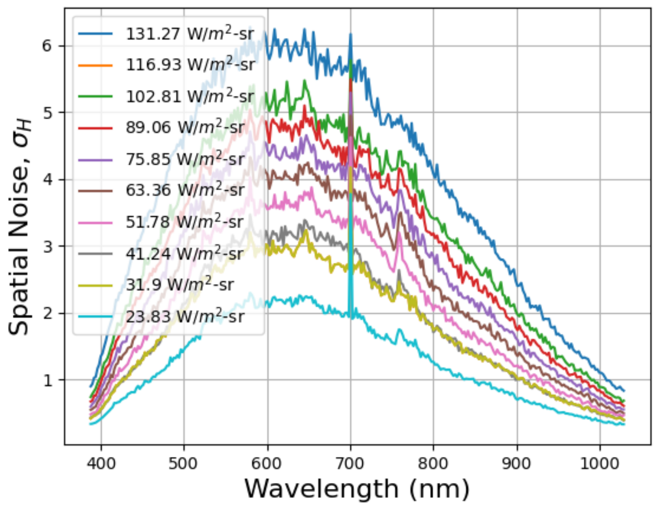

Figure 13. Spatial noise in the data cubes after applying the non-uniformity correction

Modulation Transfer Function

An image can be modeled as:

$$i(x, y) = o(x, y) \circledast h(x, y)\,, \tag{19}$$

where i(x,y) is the formed image, o(x,y) is the scene being imaged, h(x, y) is the impulse response, and$\circledast$ is the convolution operator.[26] The impulse response of a system is key in analyzing any system. Knowing the impulse response of a linear time invariant system allows us to know how the system will responds to any frequency,in this case spatial frequency. An impulse function is defined as:

$$\delta(x) = \begin{cases} 0, & \text{if } x \neq 0 \\ \infty, & \text{if } x = 0 \end{cases} \quad (20)$$

Creating an impulse in the real world can be tricky and is susceptible to erroneous results. There is no real world source that can perfectly produce an infinitely small and infinitely bright (or hot if working in emissivity space) object for a camera to view. A small pinhole could be used, but it also introduces error as it is not a perfect impulse and it falls on a single detector which is a small sample size and is subject to the response of that single detector and not the focal plane as a whole.

For this reason, the modulation transfer function (MTF) of an imager is often used as a figure of merit for an imager as it is the magnitude of the Fourier transform of the impulse response of a system. It combines the merits for resolution and modulation, or contrast, which yields a figure of merit that allows comparisons to be made from sensor to sensor.[27] In figure 14, A target with a certain spatial resolution is presented the imager, which is the perfect bar target on the left, and it is shown what the output of the system is given this spatial frequency, the blurred image on the right. This image captures the idea of what it means to transfer contrast. In the real world the contrast is 100%, but the system can only convey 20% contrast because of the resolution of the system.

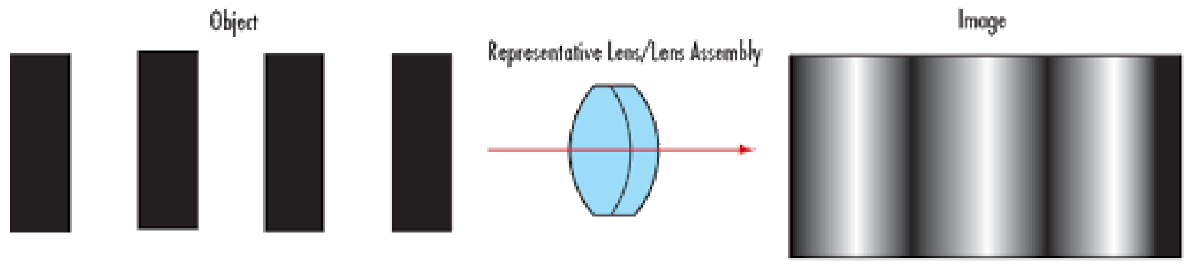

Figure 14. This figure depicts how a system transfers the contrast of a three bar target[27]

A simple way to measure the MTF of a system is to take an image of an edge as all you need is a straight edge and a light source. It is preferable to have it tilted at some angle to eliminate bias from taking data from only one row or column of a focal plane. This will yield the edge spread function (ESF) by plotting the intensity of the pixels the edge is projected on to. This is related to the step response of the system. When plotting the pixel intensities one side of the graph, say the left side, will be a high value while the other, the right side, will be lower values which creates a "step" from high digital numbers to low digital numbers. Any imaging system will blur the edge to some degree as it can only take finite samples. A step function is defined as:

$$s(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \quad (21)$$

Using the step function as the object function in equation 19, we can model the image of a knife edge as

$$i(x,y) = s(x,y) \circledast h(x,y)\,, \tag{22}$$

If a Fourier transformation of the ESF is taken it results in the equation 23

$$i(u,v) = s(u,v) * h(u,v)\,, \tag{23}$$

After the Fourier transform the convolution is now multiplication and the coordinate systems have changed. Before the Fourier transform the equations had x and y to denote spatial coordinates, and after the transform we now have u and v which denote oscillations (cycles) per unit distance. In other words, equation 23 contains the spatial frequency information of the original image.

Taking the derivative and the Fourier transform of the ESF results in the MTF.[28]

An MTF measured with the Pika-L is plotted in Figure 15.

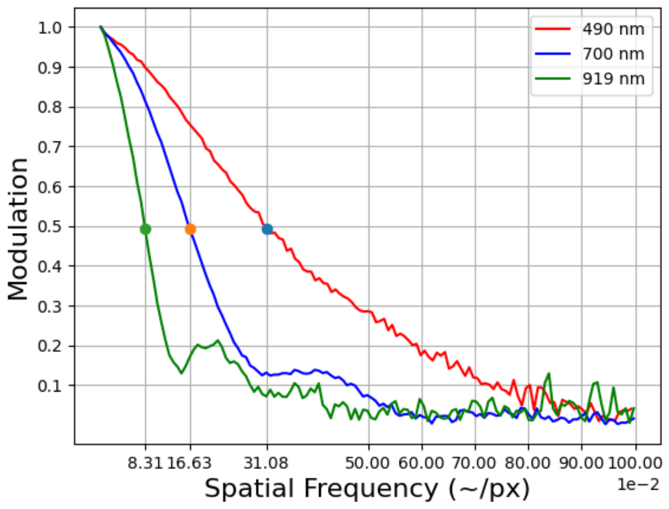

Figure 15. MTFs from three different bands. Area under the curves are 0.367, 0.204, and 0.139 from lowest wavelength to highest

The vertical axis is always between zero and one as it is the fractional modulation. The horizontal axis is spatial frequency in oscillations per pixel (∼/px). When reading this graph it can be thought of as how the contrasting power of a system decreases as the spatial frequency increases or as the number of ∼/px increases. In Figure 15 the three dots signify where each of the MTFs goes below 50% modulation. This can be a measure of the resolving power of a system, as the faster the system drops below 50 % modulation the worse the MTF generally is. The area under the MTF curve is another measure that can be used for the quality of MTFs. The higher the area under the curve the better resolving power of the system. In Figure 15, it can be seen that the 490 nm MTF has a higher modulation at most frequencies compared to the other two sampled bands. It therefore has a higher 50% modulation frequency, its being at .3108 ∼/px while the others are at 0.1663 and 0.0831 ∼/px, and a higher area under the curve as the caption mentions. The frequency at which you drop below 50% modulation can be seen as a measure of how quickly a system loses the ability to modulate frequencies and the area under the curve can be seen as a measure of how well the system modulates all frequencies.

For broadband systems, the data capture for this measurement is straight forward. A knife edge or bar target that is suitable set in front of a uniform light source is all that is needed. Then a video is taken from in front of some source to create sharp contrast. For a push broom hyperspectral sensor however it is more complicated. To create a 2D image you must move the imager or the scene. Doing this without introducing blur or any stitching issues can be a challenge, but it can be achieved by using a linear stage and calculating the distance or stage speed needed between each scan to get a full 2D image with minimal error. After collecting all of the scans, they are stitched together and the MTF for each band is calculated. The Pika-L has 300 bands therefore you need to calculate 300 MTFs which is computationally intensive.

After the data cubes are collected for the MTFs, all of the bands must be separated to load them into an MTF script that was created by David Haefner.[29] This script is an early version of the MTF tool in LabCap. After a simple script to load the images into the premade function, all values for the MTF graphs can be saved and put into a table for graphing and analysis. The main interest that was investigated was the affect of capturing a video while the imager was continuously moving versus a stop-and-go approach, referred to as the moving and step-stare methods respectively. Keeping all conditions the same, two sets of images were taken.

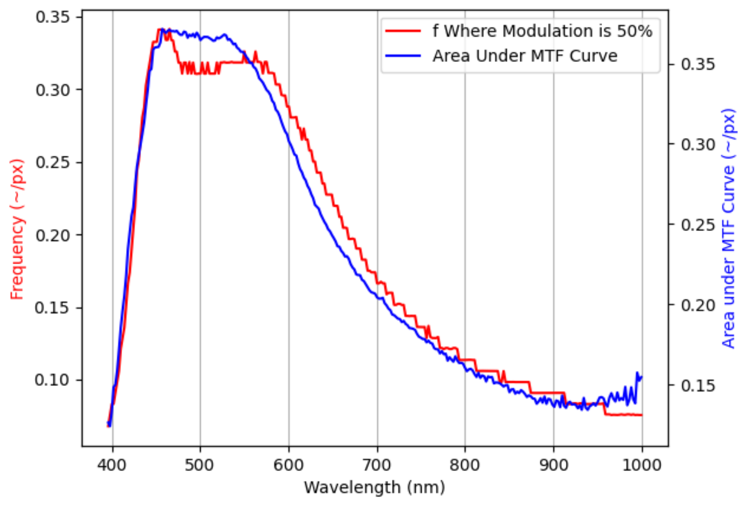

Figure 16. Area under the MTF curve and frequency where modulation is below 50% for static method

As seen in figures 16 and 17, there are minimal differences in the MTF methods. In the static method, at wavelengths around 450 to 550 nm the modulation appears to be declining more rapidly than with the static method since the 50% modulation is lower. Looking at the minimum, average, and maximum area under MTF curve, the moving method has slightly better results but not enough to definitively say it is a superior method. More data points and comparisons must be taken to make that statement.

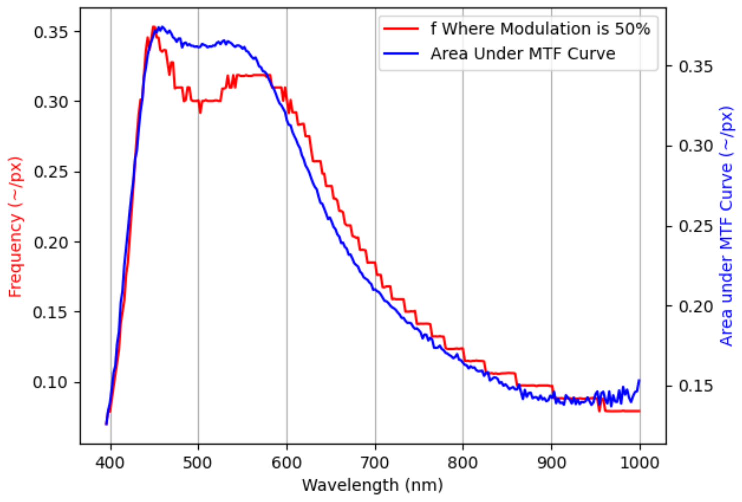

Figure 17. Area under the MTF curve and frequency where modulation is below 50% for moving method

Overall figures 16 and 17 show that MTF is worse at the low and high end of the spectral range of the Pika.This is to be expected as the bands with the best MTFs are also the bands with the highest spectral response meaning they have better signal to noise ratios and better contrast between the bright side of the knife edge and the dark side of the knife edge.

NVLabCap

NVLabCap is a program created by the Advanced Sensor Evaluation Facility (ASEF) from the U.S. Army night vision lab. It was created to be used as a tool to characterize and calibrate various sensors being developed for the military. It includes modules for MTF, field of view (FOV), 3D noise, and other measurements not used in this work. It was developed with IR and visible cameras in mind. With some work, hyperspectral imagery can be input to LabCap and give a meaningful result without developing a whole new program to do these calculations. To get it to work one must treat the spectral bands as separate cameras.

For 3D noise, there may be some way to access the FPA that is attached to the HSI and calculate noise from there, but it easier to conceptualize making each band its own image. For this approach, a break down of the data cube into its bands, which create a 2D image for each band from the spatial pixels and temporal scans, is what is needed. This will still yield the noise for each band and can even do a version of 3D noise. If you treat a scan as a 2D image and subsequent scans as frames you can obtain 3D noise statistics. The only caveat is that the row noise will always be zero as you only have a nx1 image.

For MTF measurements, separation of the bands from each other is required, as was described for the noise measurements above. The images can be input into LabCap either manually or using an executable version that is available from the ASEF's Dr. David Haefner.[29] LabCap provides a convenient way to calculate MTF once the data has been formatted for the program.

5. CONCLUSION

This thesis has covered what hyperspectral imaging is, various hyperspectral architectures, methodologies for completing noise characterization, radiometric calibration, and the measurement of the modulation transfer function of a VNIR hyperspectral imaging system. The goal of this thesis was to outline methodologies and analysis done to characterize and calibrate HSI systems in a way that is easy for other people to do these measurements themselves. Another goal was to adapt widely used broadband sensor techniques to HSI, but this seems to not work very well in the real world. 3D noise cannot be fully utilized and MTF is being moved away from in the P4001 group in favor of point spread functions from a slit target. In the future, an additional characterization to be preformed is to examine the stability and repeatability of these procedures to quantify variability. That entails a much longer study of each system and was not in the scope of this initial look at characterization. Other measurements that need to be examined are smile, keystone, wavelength accuracy, and spectral resolution. Bad pixels were also not accounted for which could casue bias in the results. There is not clear guidance in the literature on what constitutes a bad pixel. These are also important and helpful measurements to characterize a system and will be examined in future work.

Appendix

APPENDIX A. CODE USED FOR ANALYSIS

Most of the code is Python with one script being in Matlab. Also it should be noted for much faster calculations put the data cubes into numpy arrays first and then the processing goes from 20 minutes to less than one minute in the case of the noise calculations. I wanted to keep the code as it was when I produced the results so I did not make these improvements or generalizations for other users.

A Radiometer Calibration Code

This code was used to make a fit from amps to radiance for the radiometer in the integrating sphere. This step is not needed however, as you can manually compare the observed amps when taking data to the observed amps when the bulb was calibrated.

```
import numpy as np
import xlrd
from scipy import integrate, stats
import matplotlib.pyplot as plt

# load all brightness for a single bulb configuration
book_100 = xlrd.open_workbook(r'filepath\radiance_ground_truth_brightness1.xls')
index_100 = book_100.sheet_by_index(0)
aperture_100 = np.zeros((2051,)).astype(float)
x = np.zeros((2051,))
for i in range(2051):
    aperture_100[i] = index_100.cell_value(i+2, 1)
    x[i] = index_100.cell_value(i+2, 0)

# Use all of the spectral radiances in the file
book_785 = xlrd.open_workbook(r'filepath\radiance_ground_truth_brightness2.xls')
index_785 = book_785.sheet_by_index(0)
aperture_785 = np.zeros((2051,)).astype(float)
for i in range(2051):
    aperture_785[i] = index_785.cell_value(i+2, 1)
```

```
book_617 = xlrd.open_workbook(r'filepath\radiance_ground_truth_brightness3.xls')
index_617 = book_617.sheet_by_index(0)
aperture_617 = np.zeros((2051,)).astype(float)
for i in range(2051):
    aperture_617[i] = index_617.cell_value(i + 2, 1)

book_459 = xlrd.open_workbook(r'filepath\radiance_ground_truth_brightness4.xls')
index_459 = book_459.sheet_by_index(0)
aperture_459 = np.zeros((2051,)).astype(float)
for i in range(2051):
    aperture_459[i] = index_459.cell_value(i + 2, 1)

y1 = np.zeros((4,))
# Find the radiance in the sphere for each video by integrating
y1[0] = integrate.simpson(aperture_100, x)
y1[1] = integrate.simpson(aperture_785, x)
y1[2] = integrate.simpson(aperture_617, x)
y1[3] = integrate.simpson(aperture_459, x)
# multiply by 1000 becasue data is in per nm and we want per micron
y1 *= 1000
y2 = [amps1, amps2, amps3, amps4]
# find fit for amps to radiance in the sphere
a = np.polyfit(y2, y1, 1)
print(a)
print(x)

# calculate R^2
slope, intercept, r_value, p_value, std_err = stats.linregress(y1, y2)
plt.plot(y2, y1, "ob")

# plot the fit with discrete data points and R^2
plt.plot(slope * y1 + intercept, y1)
plt.xlabel('Detector Current (A)', fontsize=16)
plt.ylabel(r'Radiance ($\frac{\mu W}{cm^{2}-sr}$)', fontsize = 16)
plt.ticklabel_format(style = 'sci', axis = 'x', scilimits=(-3, -3))
plt.annotate("$R^{2} = $" + str(r_value), (.15e-3, 3e7))
plt.grid()
plt.show()
```

B Radiometric Calibration Code

This code is for making the radiance fit from digital number for the Pika-L. It outputs a csv with the scalar gain coefficient per detector.

```
import numpy as np
from spectral import *
import csv
import xlrd
from scipy import integrate

# open Labsphere calibration file, this is the ground truth data
book_100 = xlrd.open_workbook(r'filepath\bulb_ground_truth_spectral_radiance.xls')
index_100 = book_100.sheet_by_index(0)
aperture_100 = np.zeros((652,)).astype(float)
x = np.zeros((652,))
for i in range(652):
    # select only wavelengths for your camera's range
    aperture_100[i] = index_100.cell_value(i+40, 1)
    x[i] = index_100.cell_value(i+40, 0)
# this yields the radiance
radiance = integrate.simpson(aperture_100, x)
# select exact range of the camera. This makes the calibration more accurate
radiance = radiance - x[0]*.975 - x[651]*.24

# List of band center for the camera
wavelength = [390.0, 392.05, 394.1, 396.15, 398.2, 400.25, 402.3, 404.36, 406.4, 408.46,
410.52, 412.57, 414.62, 416.68, 418.74, 420.8, 422.86, 424.92, 426.98, 429.04, 431.1,
433.17, 435.24, 437.3, 439.36, 441.44, 443.5, 445.57, 447.64, 449.71, 451.78, 453.85,
455.92, 458.0, 460.07, 462.14, 464.22, 466.3, 468.37, 470.45, 472.53, 474.61, 476.69,
478.77, 480.85, 482.94, 485.02, 487.1, 489.19, 491.28, 493.36, 495.45, 497.54, 499.62,
501.72, 503.8, 505.9, 507.99, 510.08, 512.17, 514.27, 516.36, 518.46, 520.56, 522.66,
524.76, 526.86, 528.95, 531.06, 533.16, 535.26, 537.36, 539.46, 541.57, 543.67, 545.78,
547.89, 550.0, 552.1, 554.22, 556.33, 558.44, 560.55, 562.66, 564.78, 566.89, 569.01,
571.12, 573.24, 575.36, 577.48, 579.59, 581.71, 583.84, 585.96, 588.08, 590.2, 592.33,
594.45, 596.58, 598.7, 600.83, 602.96, 605.08, 607.21, 609.34, 611.48, 613.61, 615.74,
617.88, 620.01, 622.14, 624.28, 626.42, 628.56, 630.69, 632.84, 634.98, 637.12, 639.26,
641.4, 643.54, 645.68, 647.83, 649.97, 652.12, 654.26, 656.41, 658.56, 660.71, 662.86,
665.01, 667.16, 669.32, 671.47, 673.62, 675.78, 677.93, 680.09, 682.25, 684.4, 686.56,
688.72, 690.88, 693.04, 695.21, 697.37, 699.53, 701.7, 703.86, 706.03, 708.2, 710.36,
712.53, 714.7, 716.87, 719.04, 721.21, 723.38, 725.56, 727.74, 729.9, 732.08, 734.26,
736.44, 738.62, 740.8, 742.97, 745.16, 747.34, 749.52, 751.7, 753.88, 756.06, 758.25,
760.44, 762.62, 764.81, 767.0, 769.18, 771.38, 773.56, 775.76, 777.95, 780.14, 782.34,
784.53, 786.73, 788.92, 791.12, 793.32, 795.52, 797.72, 799.92, 802.12, 804.32, 806.52,
808.72, 810.93, 813.13, 815.34, 817.54, 819.75, 821.96, 824.17, 826.38, 828.59, 830.8,
833.02, 835.22, 837.44, 839.66, 841.86, 844.08, 846.3, 848.52, 850.73, 852.96, 855.18,
857.4, 859.62, 861.84, 864.06, 866.29, 868.52, 870.73, 872.96, 875.19, 877.42, 879.64,
881.88, 884.1, 886.34, 888.57, 890.8, 893.04, 895.27, 897.5, 899.74, 901.98, 904.21,
906.45, 908.69, 910.93, 913.17, 915.41, 917.66, 919.9, 922.14, 924.39, 926.63, 928.88,
931.12, 933.38, 935.62, 937.87, 940.12, 942.37, 944.62, 946.88, 949.13, 951.38, 953.64,
955.9, 958.15, 960.4, 962.66, 964.92, 967.18, 969.44, 971.7, 973.96, 976.23, 978.49,
980.76, 983.03, 985.29, 987.56, 989.82, 992.09, 994.36, 996.63, 998.9, 1001.18, 1003.44,
1005.72, 1007.99, 1010.27, 1012.54, 1014.82, 1017.1, 1019.37, 1021.65, 1023.93, 1026.21,
1028.49, 1030.78, 1033.06, 1035.34, 1037.62]

# create ground truth at Pika's band centers
spectral_radiance_interp = np.interp(wavelength, x, aperture_100)

def single(image):         # returns (900, 300) single scan, averages time
    img = image
    scans, pixels, bands = img.shape
    scan = np.zeros((pixels, bands))
    for i in range(scans):
        for j in range(pixels):
            scan[j] += img[i, j]
    scan = scan/scans
    return scan


# load different radiance videos for the fit
img0 = open_image(r'filepath\brightness1.bil.hdr')
img1 = open_image(r'filepath\brightness2.bil.hdr')
img2 = open_image(r'filepath\brightness3.bil.hdr')
img3 = open_image(r'filepath\brightness4.bil.hdr')
img4 = open_image(r'filepath\brightness5.bil.hdr')
img5 = open_image(r'filepath\brightness6.bil.hdr')
img6 = open_image(r'filepath\brightness7.bil.hdr')
img7 = open_image(r'filepath\brightness8.bil.hdr')
img8 = open_image(r'filepath\brightness9.bil.hdr')
img9 = open_image(r'filepath\brightness10.bil.hdr')
gain = open_image(r'filepath\gain.bil.hdr')     # gain header from .icp from Resonon

cubes = [img0, img1, img2, img3, img4, img5, img6, img7, img8, img9, img10]
 # list of radiances measurement was taken at
radiances = [rad1, rad2, rad3, rad4, rad5, rad6, rad7, rad8, rad9, rad10]
for i in range(len(radiances)):
    radiances[i] = radiances[i]
spectral = np.zeros((len(radiances), 300))
# scale labsphere known data to observed when taking data
for i in range(len(radiances)):
    spectral[i] = radiances[i]*spectral_radiance_interp/radiance

singles = np.zeros((len(cubes), 900, 300))
doubles = np.zeros((len(cubes), 300))
```

```
fits_single = np.zeros((900*300, 2))
fits_double = np.zeros((300, 2))
# *******************************************************
for x in range(len(cubes)):
# Take time average of all cubes, see function above
    singles[x] = single(cubes[x])
for x in range(900):
    for y in range(300):
        temp = np.zeros((len(radiances),))
        temp2 = np.zeros((len(cubes),))
        for z in range(len(cubes)):
            temp[z] = singles[z, x, y]
            temp2[z] = spectral[z, y]
# Find coeffecient to scale digital number to spectral radiance, divide by two because of
# spectral binning in Pika
        fits_single[300*x+y] = np.polyfit(temp, temp2, 1)/2
temp = np.zeros((900, 300))
for i in range(900):
    for j in range(300):
    # pull only scaling coeffecient
        temp[i, j] = fits_single[300*i+j, 0]
with open(r'filepath\Radiance_Cal.csv', "w", newline='') as myfile:
    writer = csv.writer(myfile, quoting=csv.QUOTE_ALL)
    for i in range(300):
    # save data to csv, will overwrite if already exists
        writer.writerow(temp[:, i])
```

C Radiance Calibration Verification Code

This is the code that was used to measure the percent error between the theoretical known spectral radiance in the sphere and the corrected Pika output.

```
import numpy as np
from spectral import *
```

```
import csv
import matplotlib.pyplot as plt
import xlrd
from scipy import integrate

wavelength = [390.0, 392.05, 394.1, 396.15, 398.2, 400.25, 402.3, 404.36, 406.4, 408.46,
410.52, 412.57, 414.62, 416.68, 418.74, 420.8, 422.86, 424.92, 426.98, 429.04, 431.1,
433.17, 435.24, 437.3, 439.36, 441.44, 443.5, 445.57, 447.64, 449.71, 451.78, 453.85,
455.92, 458.0, 460.07, 462.14, 464.22, 466.3, 468.37, 470.45, 472.53, 474.61, 476.69,
478.77, 480.85, 482.94, 485.02, 487.1, 489.19, 491.28, 493.36, 495.45, 497.54, 499.62,
501.72, 503.8, 505.9, 507.99, 510.08, 512.17, 514.27, 516.36, 518.46, 520.56, 522.66,
524.76, 526.86, 528.95, 531.06, 533.16, 535.26, 537.36, 539.46, 541.57, 543.67, 545.78,
547.89, 550.0, 552.1, 554.22, 556.33, 558.44, 560.55, 562.66, 564.78, 566.89, 569.01,
571.12, 573.24, 575.36, 577.48, 579.59, 581.71, 583.84, 585.96, 588.08, 590.2, 592.33,
594.45, 596.58, 598.7, 600.83, 602.96, 605.08, 607.21, 609.34, 611.48, 613.61, 615.74,
617.88, 620.01, 622.14, 624.28, 626.42, 628.56, 630.69, 632.84, 634.98, 637.12, 639.26,
641.4, 643.54, 645.68, 647.83, 649.97, 652.12, 654.26, 656.41, 658.56, 660.71, 662.86,
665.01, 667.16, 669.32, 671.47, 673.62, 675.78, 677.93, 680.09, 682.25, 684.4, 686.56,
688.72, 690.88, 693.04, 695.21, 697.37, 699.53, 701.7, 703.86, 706.03, 708.2, 710.36,
712.53, 714.7, 716.87, 719.04, 721.21, 723.38, 725.56, 727.74, 729.9, 732.08, 734.26,
736.44, 738.62, 740.8, 742.97, 745.16, 747.34, 749.52, 751.7, 753.88, 756.06, 758.25,
760.44, 762.62, 764.81, 767.0, 769.18, 771.38, 773.56, 775.76, 777.95, 780.14, 782.34,
784.53, 786.73, 788.92, 791.12, 793.32, 795.52, 797.72, 799.92, 802.12, 804.32, 806.52,
808.72, 810.93, 813.13, 815.34, 817.54, 819.75, 821.96, 824.17, 826.38, 828.59, 830.8,
833.02, 835.22, 837.44, 839.66, 841.86, 844.08, 846.3, 848.52, 850.73, 852.96, 855.18,
857.4, 859.62, 861.84, 864.06, 866.29, 868.52, 870.73, 872.96, 875.19, 877.42, 879.64,
881.88, 884.1, 886.34, 888.57, 890.8, 893.04, 895.27, 897.5, 899.74, 901.98, 904.21,
906.45, 908.69, 910.93, 913.17, 915.41, 917.66, 919.9, 922.14, 924.39, 926.63, 928.88,
931.12, 933.38, 935.62, 937.87, 940.12, 942.37, 944.62, 946.88, 949.13, 951.38, 953.64,
955.9, 958.15, 960.4, 962.66, 964.92, 967.18, 969.44, 971.7, 973.96, 976.23, 978.49,
980.76, 983.03, 985.29, 987.56, 989.82, 992.09, 994.36, 996.63, 998.9, 1001.18, 1003.44,
1005.72, 1007.99, 1010.27, 1012.54, 1014.82, 1017.1, 1019.37, 1021.65, 1023.93, 1026.21,
1028.49, 1030.78, 1033.06, 1035.34, 1037.62]
```

```
# load bulb spectral curve
book_100 = xlrd.open_workbook(r'filepath\spectral_curve.xls')
index_100 = book_100.sheet_by_index(0)
aperture_100 = np.zeros((2050,)).astype(float)
x = np.zeros((2050,))
for i in range(2050):
    aperture_100[i] = index_100.cell_value(i+2, 1)*1000    # *1000 for per nm to per um
    x[i] = index_100.cell_value(i+2, 0)
# make array of spectral radiance at Pika's bandcenters
binned_labpshere = np.interp(wavelength, x, aperture_100)

#open cube of bulb that was not used to make the radiance calibration
img0 = open_image(r'filepath\bulb_b.bil.hdr')
fits = np.zeros((900, 300))
mine = np.zeros((900, 300))
# load radiance coeffecients
with open(r'filepath\Radiance_Cal.csv', "r", newline='') as myfile:
    reader = csv.reader(myfile, quoting=csv.QUOTE_ALL)
    i = 0
    for row in reader:
        fits[:, i] = row
        i += 1
# apply the radiance coeffecients
for i in range(900):
    for j in range(300):
        mine[i, j] = fits[i, j]*img0[0, i, j]



mine_avg = np.zeros((300, ))
for i in range(300):
    # make sure gain and integration time are accounted for
    # for calculation look at Resonon source code for Spectronon
    mine_avg[i] = np.average(mine[:, i])

# calculate percent error between coreected data and theoretical
my_error = (binned_labpshere-mine_avg)/binned_labpshere
my_error = abs(my_error)*100
plt.plot(wavelength, my_error, label = 'Mine')
print(np.min(my_error), np.average(my_error), np.max(my_error))

# plt.plot(wavelength, mine_avg, label = 'me')
# plt.plot(x, aperture_100, label = 'theoretical')
plt.ylabel("Radiance Calibration Error (%) ", fontsize=16)
plt.xlabel("Wavelength (nm)", fontsize=16)
plt.legend()
plt.grid()
plt.show()
```

D Noise Calculation Code

This is the code for calculating all four noise components (total, temporal, spatial, temporal-spatial)

```
import numpy as np
from spectral import *
import matplotlib.pyplot as plt

# load flat field data
img0 = open_image(r'filepath\brightness1.bil.hdr')
img1 = open_image(r'filepath\brightness1.bil.hdr')
img2 = open_image(r'filepath\brightness1.bil.hdr')
img3 = open_image(r'filepath\brightness1.bil.hdr')
img4 = open_image(r'filepath\brightness1.bil.hdr')
img5 = open_image(r'filepath\brightness1.bil.hdr')
img6 = open_image(r'filepath\brightness1.bil.hdr')
img7 = open_image(r'filepath\brightness1.bil.hdr')
img8 = open_image(r'filepath\brightness1.bil.hdr')
```

```
img9 = open_image(r'filepath\brightness1.bil.hdr')
scans, pixels, bands = img0.shape
cubes = [img0, img1, img2, img3, img4, img5, img6, img7, img8, img9]

# get sigma_th and global average for each band, take standard deviation and mean of
# spatial and temporal samples per band
sigma_th = np.zeros((len(cubes), bands))
avg = np.zeros((len(cubes), bands))
for j in range(len(cubes)):
    img = cubes[j]
    for i in range(bands):
        avg[j, i] = np.average(img[:, :, i])
        sigma_th[j, i] = (np.std(img[:, :, i]-avg[j, i]))


# Get sigma_h, time average the data then take standard deviation
# t_avg = np.zeros((len(cubes), pixels, bands)).astype(float)
# for j in range(len(cubes)):
#     img = cubes[j]
#     for i in range(pixels):
#         print(j, i)
#         for k in range(bands):
#             t_avg[j, i, k] = np.average(img[:, i, k]-avg[j, k])
# sigma_h = np.zeros((len(cubes), bands))
# for j in range(len(cubes)):
#     img = cubes[j]
#     for i in range(bands):
#         sigma_h[j, i] = np.std(t_avg[j, :, i])/avg[j, i]


#
# # Get sigma_t, spatially average the data then take a standarad deviation
# s_avg = np.zeros((len(cubes), scans, bands)).astype(float)
# for k in range(len(cubes)):
#     img = cubes[k]
#     for i in range(scans):
#         print(k, i)
#         for j in range(bands):
#             s_avg[k, i, j] = np.average(img[i, :, j]-avg[k, j])
# sigma_t = np.zeros((len(cubes), bands))
# for i in range(bands):
#     for j in range(len(cubes)):
#         sigma_t[j, i] = np.std(s_avg[j, :, i])/avg[j, i]



print(np.min(sigma_th), np.max(sigma_th), np.average(sigma_th))
      # np.min(sigma_t), np.max(sigma_t), np.average(sigma_t),
      # np.min(sigma_h), np.max(sigma_h), np.average(sigma_h))

averages = np.zeros((len(cubes), ))

# for i in range(len(cubes)):
#     averages[i] = np.average(sigma_h[i, :])

radiances = [23.83, 31.90, 41.24, 51.78, 63.36, 75.85, 89.06, 102.81, 116.93, 131.27]
wavelength = [390.0, 392.05, 394.1, 396.15, 398.2, 400.25, 402.3, 404.36, 406.4, 408.46,
410.52, 412.57, 414.62, 416.68, 418.74, 420.8, 422.86, 424.92, 426.98, 429.04, 431.1,
433.17, 435.24, 437.3, 439.36, 441.44, 443.5, 445.57, 447.64, 449.71, 451.78, 453.85,
455.92, 458.0, 460.07, 462.14, 464.22, 466.3, 468.37, 470.45, 472.53, 474.61, 476.69,
478.77, 480.85, 482.94, 485.02, 487.1, 489.19, 491.28, 493.36, 495.45, 497.54, 499.62,
501.72, 503.8, 505.9, 507.99, 510.08, 512.17, 514.27, 516.36, 518.46, 520.56, 522.66,
524.76, 526.86, 528.95, 531.06, 533.16, 535.26, 537.36, 539.46, 541.57, 543.67, 545.78,
547.89, 550.0, 552.1, 554.22, 556.33, 558.44, 560.55, 562.66, 564.78, 566.89, 569.01,
571.12, 573.24, 575.36, 577.48, 579.59, 581.71, 583.84, 585.96, 588.08, 590.2, 592.33,
594.45, 596.58, 598.7, 600.83, 602.96, 605.08, 607.21, 609.34, 611.48, 613.61, 615.74,
617.88, 620.01, 622.14, 624.28, 626.42, 628.56, 630.69, 632.84, 634.98, 637.12, 639.26,
641.4, 643.54, 645.68, 647.83, 649.97, 652.12, 654.26, 656.41, 658.56, 660.71, 662.86,
665.01, 667.16, 669.32, 671.47, 673.62, 675.78, 677.93, 680.09, 682.25, 684.4, 686.56,
688.72, 690.88, 693.04, 695.21, 697.37, 699.53, 701.7, 703.86, 706.03, 708.2, 710.36,
712.53, 714.7, 716.87, 719.04, 721.21, 723.38, 725.56, 727.74, 729.9, 732.08, 734.26,
736.44, 738.62, 740.8, 742.97, 745.16, 747.34, 749.52, 751.7, 753.88, 756.06, 758.25,
760.44, 762.62, 764.81, 767.0, 769.18, 771.38, 773.56, 775.76, 777.95, 780.14, 782.34,
784.53, 786.73, 788.92, 791.12, 793.32, 795.52, 797.72, 799.92, 802.12, 804.32, 806.52,
808.72, 810.93, 813.13, 815.34, 817.54, 819.75, 821.96, 824.17, 826.38, 828.59, 830.8,
833.02, 835.22, 837.44, 839.66, 841.86, 844.08, 846.3, 848.52, 850.73, 852.96, 855.18,
857.4, 859.62, 861.84, 864.06, 866.29, 868.52, 870.73, 872.96, 875.19, 877.42, 879.64,
881.88, 884.1, 886.34, 888.57, 890.8, 893.04, 895.27, 897.5, 899.74, 901.98, 904.21,
906.45, 908.69, 910.93, 913.17, 915.41, 917.66, 919.9, 922.14, 924.39, 926.63, 928.88,
931.12, 933.38, 935.62, 937.87, 940.12, 942.37, 944.62, 946.88, 949.13, 951.38, 953.64,
955.9, 958.15, 960.4, 962.66, 964.92, 967.18, 969.44, 971.7, 973.96, 976.23, 978.49,
980.76, 983.03, 985.29, 987.56, 989.82, 992.09, 994.36, 996.63, 998.9, 1001.18, 1003.44,
1005.72, 1007.99, 1010.27, 1012.54, 1014.82, 1017.1, 1019.37, 1021.65, 1023.93, 1026.21,
1028.49, 1030.78, 1033.06, 1035.34, 1037.62]

# calculate sigma th which is sqrt(total^2 - temporal^2 - spatial^2)
sigma = np.zeros((len(cubes), bands))
for i in range(bands):
     for j in range(len(cubes)):
         sigma[j, i] = np.sqrt(sigma_th[j, i]*sigma_th[j, i]-sigma_t[j, i]*sigma_t[j, i]
            -sigma_h[j, i]*sigma_h[j, i])

# the rest is graphing functions for the noise
x = np.arange(0, len(cubes), 1, dtype= int)
x_10 = np.zeros((len(cubes), bands))
for i in range(len(cubes)):
    x_10[i] = x[i]
# print(np.min(sigma), np.max(sigma), np.average(sigma))
# j = 9
# for i in range(len(cubes)):
```

```
#       # plt.errorbar(wavelength, avg[j, :], yerr = sigma_th[j], label = str(radiances[j])+
    # " W/$m^2$-sr")
#       plt.plot(wavelength, sigma_th[j, :], label = str(radiances[j])+" W/$m^2$-sr")
#       j = j-1
# plt.xlabel("Wavelength (nm)", fontsize=16)
# plt.ylabel("Total Noise, $\sigma_{total}$", fontsize=16)
# plt.legend(loc = "upper right")
# plt.grid()
# plt.show()
#
# j = 9
# for i in range(len(cubes)):
#       # plt.errorbar(wavelength, avg[j, :], yerr = sigma_h[j], label = str(radiances[j])
+" W/$m^2$-sr")
#       plt.plot(wavelength, sigma_h[j, :], label = str(radiances[j])+" W/$m^2$-sr")
#       j = j-1
# plt.xlabel("Wavelength (nm)", fontsize=16)
# plt.ylabel("Spatial Noise, $\sigma_{H}$", fontsize=16)
# plt.legend(loc = "upper right")
# plt.grid()
# plt.show()
#
# j = len(cubes) - 1
# for i in range(len(cubes)):
#       # plt.errorbar(wavelength, avg[j, :], yerr = sigma_t[j], label = str(radiances[j])
+" W/$m^2$-sr")
#       plt.plot(wavelength, sigma_t[j, :], label = str(radiances[j])+" W/$m^2$-sr")
#       j = j-1
# plt.xlabel("Wavelength (nm)", fontsize=16)
# plt.ylabel("Temporal Noise, $\sigma_{T}$", fontsize=16)
# plt.legend(loc = "upper right")
# plt.grid()
# plt.show()
# j = len(cubes) - 1
# for i in range(len(cubes)):
#     # plt.errorbar(wavelength, avg[j, :], yerr = sigma_t[j], label = str(radiances[j])
+" W/$m^2$-sr")
#     plt.plot(wavelength, sigma[j, :], label = str(radiances[j])+" W/$m^2$-sr")
#     j = j-1
# plt.xlabel("Wavelength (nm)", fontsize=16)
# plt.ylabel("Temporal Column Noise, $\sigma_{TH}$", fontsize=16)
# plt.legend(loc = "upper right")
# plt.grid()
# plt.show()

plt.plot(avg[:, 150], sigma_th[:, 150])
plt.xlabel("Wavelength (nm)", fontsize=16)
plt.ylabel("Total Noise, $\sigma_{total}$", fontsize=16)
plt.grid()
plt.show()
```

E NUC Code

This code is for making a NUC file for the Pika.

```
import numpy as np
from spectral import *
import csv
import matplotlib.pyplot as plt

# load same data used for radiance calibration and noise
img0 = open_image(r'filepath\brightness1.bil.hdr')
img1 = open_image(r'filepath\brightness2.bil.hdr')
img2 = open_image(r'filepath\brightness3.bil.hdr')
img3 = open_image(r'filepath\brightness4.bil.hdr')
img4 = open_image(r'filepath\brightness5.bil.hdr')
```

```
img5 = open_image(r'filepath\brightness6.bil.hdr')
img6 = open_image(r'filepath\brightness7.bil.hdr')
img7 = open_image(r'filepath\brightness8.bil.hdr')
img8 = open_image(r'filepath\brightness9.bil.hdr')
img9 = open_image(r'filepath\brightness10.bil.hdr')

scans, pixels, bands = img1.shape
# use this line for checking spatial noise after NUC
# cubes = [img0, img1, img2, img3, img4, img5, img6, img7, img8, img9]
cubes = [img1, img8]



avg = np.zeros((2, 300))
for i in range(len(cubes)):
    img = cubes[i]
    for j in range(bands):
        avg[i, j] = np.average(img[:, :, j])            # global average for each band

t_avg = np.zeros((2, pixels, bands))
for j in range(len(cubes)):
    img = cubes[j]
    for i in range(pixels):
        print(j, i)
        for k in range(bands):
        # temporal average for each pixel
            t_avg[j, i, k] = np.average(img[:, i, k])

fits = np.zeros((270000, 2))
for i in range(pixels):
# Take each pixel equal the global average for the band
    for j in range(bands):
        fits[300*i+j] = np.polyfit(t_avg[:, i, j], avg[:, j], 1)
```

```
with open(r'filepath\NUC.csv', "w", newline='') as myfile:     # save to csv
    writer = csv.writer(myfile, quoting=csv.QUOTE_ALL)
    for i in range(len(fits)):
        writer.writerow((fits[i, 0], fits[i, 1]))

# The rest of the code is for inspecting the noise after a NUC, see noise code for details
# fits = np.zeros((270000, 2))
# with open(r'filepath\NUC.csv', "r", newline='') as myfile:
#     reader = csv.reader(myfile, quoting=csv.QUOTE_ALL)
#     i = 0
#     for row in reader:
#         fits[i] = row
#         i += 1
#
# t_avg = np.zeros((10, pixels, bands)).astype(float)
# for j in range(len(cubes)):
#     img = cubes[j]
#     for i in range(pixels):
#         print(j, i)
#         for k in range(bands):
#             t_avg[j, i, k] = np.average(img[:, i, k])
#
# videos = np.zeros((10, pixels, bands))
# for k in range(len(cubes)):
#     img = np.zeros((1, pixels, bands))
#     for i in range(pixels):
#         for j in range(bands):
#             img[0, i, j] = t_avg[k, i, j]*fits[300*i+j, 0] + fits[300*i+j, 0]
#     videos[k] = img
#
#
# sigma_h = np.zeros((10, 300))
# for j in range(len(cubes)):
```

```
#      img = videos[j]
#      for i in range(bands):
#          sigma_h[j, i] = np.std(img[:, i])
#
# print(np.min(sigma_h), np.max(sigma_h), np.average(sigma_h))
#
# radiances = [23.83, 31.90, 41.24, 51.78, 63.36, 75.85, 89.06, 102.81, 116.93, 131.27]
wavelength = [390.0, 392.05, 394.1, 396.15, 398.2, 400.25, 402.3, 404.36, 406.4, 408.46,
410.52, 412.57, 414.62, 416.68, 418.74, 420.8, 422.86, 424.92, 426.98, 429.04, 431.1,
433.17, 435.24, 437.3, 439.36, 441.44, 443.5, 445.57, 447.64, 449.71, 451.78, 453.85,
455.92, 458.0, 460.07, 462.14, 464.22, 466.3, 468.37, 470.45, 472.53, 474.61, 476.69,
478.77, 480.85, 482.94, 485.02, 487.1, 489.19, 491.28, 493.36, 495.45, 497.54, 499.62,
501.72, 503.8, 505.9, 507.99, 510.08, 512.17, 514.27, 516.36, 518.46, 520.56, 522.66,
524.76, 526.86, 528.95, 531.06, 533.16, 535.26, 537.36, 539.46, 541.57, 543.67, 545.78,
547.89, 550.0, 552.1, 554.22, 556.33, 558.44, 560.55, 562.66, 564.78, 566.89, 569.01,
571.12, 573.24, 575.36, 577.48, 579.59, 581.71, 583.84, 585.96, 588.08, 590.2, 592.33,
594.45, 596.58, 598.7, 600.83, 602.96, 605.08, 607.21, 609.34, 611.48, 613.61, 615.74,
617.88, 620.01, 622.14, 624.28, 626.42, 628.56, 630.69, 632.84, 634.98, 637.12, 639.26,
641.4, 643.54, 645.68, 647.83, 649.97, 652.12, 654.26, 656.41, 658.56, 660.71, 662.86,
665.01, 667.16, 669.32, 671.47, 673.62, 675.78, 677.93, 680.09, 682.25, 684.4, 686.56,
688.72, 690.88, 693.04, 695.21, 697.37, 699.53, 701.7, 703.86, 706.03, 708.2, 710.36,
712.53, 714.7, 716.87, 719.04, 721.21, 723.38, 725.56, 727.74, 729.9, 732.08, 734.26,
736.44, 738.62, 740.8, 742.97, 745.16, 747.34, 749.52, 751.7, 753.88, 756.06, 758.25,
760.44, 762.62, 764.81, 767.0, 769.18, 771.38, 773.56, 775.76, 777.95, 780.14, 782.34,
784.53, 786.73, 788.92, 791.12, 793.32, 795.52, 797.72, 799.92, 802.12, 804.32, 806.52,
808.72, 810.93, 813.13, 815.34, 817.54, 819.75, 821.96, 824.17, 826.38, 828.59, 830.8,
833.02, 835.22, 837.44, 839.66, 841.86, 844.08, 846.3, 848.52, 850.73, 852.96, 855.18,
857.4, 859.62, 861.84, 864.06, 866.29, 868.52, 870.73, 872.96, 875.19, 877.42, 879.64,
881.88, 884.1, 886.34, 888.57, 890.8, 893.04, 895.27, 897.5, 899.74, 901.98, 904.21,
906.45, 908.69, 910.93, 913.17, 915.41, 917.66, 919.9, 922.14, 924.39, 926.63, 928.88,
931.12, 933.38, 935.62, 937.87, 940.12, 942.37, 944.62, 946.88, 949.13, 951.38, 953.64,
955.9, 958.15, 960.4, 962.66, 964.92, 967.18, 969.44, 971.7, 973.96, 976.23, 978.49,
980.76, 983.03, 985.29, 987.56, 989.82, 992.09, 994.36, 996.63, 998.9, 1001.18, 1003.44,
1005.72, 1007.99, 1010.27, 1012.54, 1014.82, 1017.1, 1019.37, 1021.65, 1023.93, 1026.21,
1028.49, 1030.78, 1033.06, 1035.34, 1037.62]
#
# j = 9
# for i in range(10):
#     # plt.errorbar(x_10[j, :], avg[j, :], yerr = sigma_h[j], label = str(radiances[j])+
        # " W/$m^2$-sr")
#     plt.plot(wavelength, sigma_h[j, :], label = str(radiances[j])+" W/$m^2$-sr")
#     j = j-1
# plt.xlabel("Wavelength (nm)", fontsize=16)
# plt.ylabel("Spatial Noise, $\sigma_{H}$", fontsize=16)
# plt.legend(loc = "upper right")
# plt.grid()
# plt.show()
```

F Record MTF Script

This script was used to collect the step-stare MTF. It uses a Zaber linear stage which can be communicated with over serial and PyPylon which requires Pylon viewer for the Basler CMOS camera in the Pika.

```
# Most of this code is from https://resonon.gitlab.io/programming-docs/


import numpy as np
from pypylon import pylon
from zaber_motion import Units
from zaber_motion.ascii import Connection
import envi_io
import io


# Pika L Parameters, from table on the Camera Setup and Windowing Page
ROI_WIDTH = 900
ROI_HEIGHT = 600
Y_BINNING = 1
```

```
# Device-specific parameters obtained from Resonon Camera Configuration Report
X_OFFSET = 516
Y_OFFSET = 312
A = 0.0001005000012810342
B = 0.9494590163230896
C = 81.38700103759766

# obtain the camera instance from the Pylon API
camera_instance = pylon.InstantCamera(pylon.TlFactory.GetInstance().CreateFirstDevice())
camera_instance.Open()

# Configure the camera window
# We make sure the camera is not binned before configuring the window, then add our desired
# binning
camera_instance.BinningHorizontal.SetValue(1)
camera_instance.BinningVertical.SetValue(1)
camera_instance.Width.SetValue(ROI_WIDTH)
camera_instance.Height.SetValue(ROI_HEIGHT)
camera_instance.OffsetX.SetValue(X_OFFSET)
camera_instance.OffsetY.SetValue(Y_OFFSET)
camera_instance.BinningVertical.SetValue(Y_BINNING)
camera_instance.ExposureTime.SetValue(15000)
camera_instance.PixelFormat.SetValue('Mono12')
camera_instance.Gain.SetValue(0.0)



def get_wavelength_for_channel(band_number: int) -> float:
    # The wavelength calibration equation is defined in terms of un-binned and un-windowed
    # sensor coordinates.
    # here, we convert the band number of the binned and windowed region to its equivalent
    # un-binned location
    # and apply the calibration equation.
    # The term Y_BINNING / 2.0 - 0.5 is a correction to ensure the wavelength is calculatec
    #from the center of the binned region, instead of the edge
    # (using 0-based, C style indexing)
    camera_pixel = Y_OFFSET + band_number * Y_BINNING + Y_BINNING / 2.0 - 0.5
    return A * camera_pixel**2 + B * camera_pixel + C


wavelengths = [get_wavelength_for_channel(pixel) for pixel in range(camera_instance.
    Height.GetValue())]
wavelengths = np.asarray(wavelengths)
wavelengths = wavelengths.reshape((300, 2))
final = np.zeros((300, ))
final = (wavelengths[:, 0] + wavelengths[:, 1])/2
cube = envi_io.Cube.new(dtype=np.uint16, bands=300,
                        lines=900, samples=900, interleave="bil",
                        wavelength=final
                        )
with Connection.open_serial_port("COM4") as connection: # connect to zaber linear stage
    connection.enable_alerts()

    device_list = connection.detect_devices()
    print("Found {} devices".format(len(device_list)))

    device = device_list[0]

    axis = device.get_axis(1)
    if not axis.is_homed():             # send the stage to its home
        axis.home()
    axis.move_absolute(5, Units.LENGTH_MILLIMETRES)
    # Capture 900 frames
    frame_number = 0
    while frame_number < 900:
        temp = np.zeros((600, 900))
        camera_instance.StartGrabbing(1)
        grab_result = camera_instance.RetrieveResult
            (5000, pylon.TimeoutHandling_ThrowException)
        if grab_result.GrabSucceeded():
            temp = np.transpose(grab_result.Array)
            temp = temp.reshape((900, 300, 2))
            cube.data[frame_number] = np.transpose(temp[:, :, 0]) +
            np.transpose(temp[:, :, 1]) # software binning
            frame_number += 1
        grab_result.Release()
        camera_instance.StopGrabbing()
        print(frame_number)
        # need to calculate sampling distance for measurement for the distance
        axis.move_relative(0.2687, Units.LENGTH_MILLIMETRES)
camera_instance.Close()

cube.header["shutter"] = 15.0
cube.header["gain"] = 0.0
cube.header["spectral binning"] = 2
cube.header["sample binning"] = 1
cube.header["ceiling"] = 8190

cube_name = set({"gain": cube}.keys())
header_fp = io.BytesIO()
cube.write_fp(header_fp=header_fp)
header_data = header_fp.getvalue()

data_fp = io.BytesIO()
cube.write_fp(data_fp=data_fp)
data = data_fp.getvalue()

with open(r"filepath\mtf_stare_stop.bil.hdr", "wb") as h:
    # Write bytes to file
    h.write(header_data)


with open(r"filepath\mtf_stare_stop.bil", "wb") as d:
    # Write bytes to file
    d.write(data)
```

G MTF Measurement

This is the Matlab code that calls the MTF measurement method from Haefner for each band. I opened the array in matlab and copied and pasted it into excel for analysis later.

```
% This code is in Matlab since the function being called was in matlab, this is from
% https://www.mathworks.com/matlabcentral/fileexchange/66894-mtfdh?s_tid=prof_contriblnk
clear all;
x = multibandread("filepath\nuc_mtf_stat.bil", [900, 900, 300], ...     % load cube
                    'float64', 0, 'bil', 'ieee-le');
imagesc(x(:, :, 150))
img = (x(:, :, 150));
Input=[];
% General
Input.SlitFlag = 0;                     % 0 is an edge, 1 is a slit
Input.Rat = 0.995;                      % Ratio of iFOV_v/iFOV_h
Input.ReCrop=1;                         % Allow for a recrop and rotation or ROI
Input.FindSemiCirc=1;                   % Find the half moon target =1,
Input.BlackHot=0;                       % 0 is white hot, 1 is blackhot (for circ finding)
Input.PlotFigures=0;                    % 0, 1, 2
% Advanced
Input.ISOmethod=0;                      % ISO projection method or shift and sum
Input.UseHammingWin=0;                  % Apply HammingWindow = 1, Rect Window = 0
Input.OverSample = 0;                   % 0 is find optimal, 2 is dx =0.5...4 is ISO dx=0.
Input.UsePhaseCorrectedRealPart=0;      % 0
Input.MTFmetric=1;                      % Metric for MTF focus evaluation 1 is Gaussian fit
                                        % 2 is root integral of HS, 3 is specfic freq
Input.ESFFitMethod=1;                   % Fit derivative to Gaussian = 1, Fit ESF = 2
```

```
% Color
Input.WBcoeff=0;
% On Save Click
Input.SaveMTF=0;                          % Saving MTF... calcs fit
Input.CalcNoise=0;                        % Calculate noise = 1, ignore =1

% Input Params hidden
Input.CenterROI=1;                         %[1 is max, -1 is min, 0 is do not center
Input.StencilDerivativeOrder=3;           % 3 or 5 point derivative...3 is fine
Input.CropCornerProjection=0;             % Crop corner of ROI for equal averaging...
                                          % not important if recropping
Input.InterpolationKernel=1;              % See LinInterpDH... 1 is Lancos 3
Input.XdeltaThresh = 2;                   % Potential shift to allow prior to resample
Input.ApplySampCor=1;                     % Apply sampling correction...should be 1
Input.MTFmetricFreq=.25;                  % MTF Metric frequency eval
Input.NoisePvals=[.1,.9];

% Determine ROI
[~,pos]=ImRectCropDH(img);
Input.ROItgt=pos;

mtf_out = zeros(519, 3);
freq_out = zeros(3, 519);
% Call MTF calc
for i = 1:1:length(x(0, 0, :))
    % I transposed one of the matrices to make them the same size then
    % copied and pasted them into a xlsx for analysis
    MTF_DHnewRS=MTFCalcDH(x(:, :, i),Input,0*x(:, :, i));
    mtf_out(1:length(MTF_DHnewRS.MTF), 1) = abs(MTF_DHnewRS.MTF);
    freq_out(1, 1:length(MTF_DHnewRS.Freq)) = MTF_DHnewRS.Freq;
end
clf
```

H Graph MTF Measurement

This code was for analyzing the the MTF measurements. It utilizes area under the curve and MTF50. It produces graphs of both.

```
import numpy as np
import pandas as pd
from scipy import integrate
import matplotlib.pyplot as plt

# import mtf data made in appendix G
dfy = pd.read_excel(r"filepath\moving_mtf.xlsx")
dfx = pd.read_excel(r"filepath\moving_freq.xlsx")
ones = np.ones(296)
zeros = np.zeros(296)

mtf = dfy.to_numpy()
mtf = np.vstack([ones, mtf])
freq = dfx.to_numpy()
freq = np.vstack([zeros, freq])

area = np.zeros(283)
for i in range(283):
    j = 0
    tempy = np.zeros(len(freq))
    tempx = np.zeros(len(freq))
# make sure MTF is less than 1
    while freq[j, i] <= 1:
        dx = freq[1, i] - freq[0, i]
        tempy[j] = mtf[j, i]
        tempx[j] = freq[j, i]
        j += 1
    tempy = np.resize(tempy, j)
    tempx = np.resize(tempx, j)
    if i == 46:
        band_50_mtf = tempx
        freq_50_mtf = tempy
    if i == 146:
        band_150_mtf = tempx
        freq_150_mtf = tempy
    if i == 246:
        band_250_mtf = tempx
        freq_250_mtf = tempy

    # calculate area under MTF curve
    area[i] = integrate.simpson(tempy, tempx, dx)
under = np.zeros(283)
for i in range(283):
    j = 0
    tempy = np.zeros(len(freq))
    tempx = np.zeros(len(freq))
    # find at what frequency the MTF50 is
    while mtf[j, i] >= 0.5:
        j += 1
    under[i] = freq[j, i]
    if i == 46:
        freq1 = j
    if i == 146:
        freq2 = j
    if i == 246:
        freq3 = j

x = np.arange(4, 300, 1)
wavelength = [396.02, 398.04, 400.07, 402.1, 404.13, 406.16, 408.19, 410.22, 412.26, 414.3
416.33, 418.37, 420.4, 422.44, 424.48, 426.52, 428.56, 430.6, 432.64, 434.68, 436.73,
438.77, 440.82, 442.86, 444.91, 446.96, 449.0, 451.06, 453.1, 455.16, 457.21, 459.26,
461.32, 463.36, 465.42, 467.48, 469.54, 471.59, 473.64, 475.71, 477.76, 479.82, 481.88,
483.94, 486.02, 488.08, 490.14, 492.21, 494.27, 496.34, 498.4, 500.48, 502.54, 504.61,
506.68, 508.76, 510.83, 512.9, 514.98, 517.05, 519.12, 521.2, 523.28, 525.36, 527.44,
529.51, 531.6, 533.68, 535.76, 537.84, 539.92, 542.01, 544.1, 546.18, 548.27, 550.36,
552.44, 554.54, 556.62, 558.72, 560.81, 562.9, 564.99, 567.08, 569.18, 571.28, 573.38,
575.47, 577.57, 579.66, 581.76, 583.86, 585.96, 588.07, 590.17, 592.28, 594.38, 596.48,
598.6, 600.7, 602.81, 604.92, 607.02, 609.14, 611.24, 613.36, 615.47, 617.58, 619.7, 621.8
623.93, 626.05, 628.16, 630.28, 632.4, 634.52, 636.64, 638.76, 640.88, 643.01, 645.13,
647.26,649.38, 651.51, 653.64, 655.76, 657.89, 660.02, 662.15, 664.29, 666.42, 668.54,
670.68, 672.82, 674.95, 677.08, 679.22, 681.36, 683.5, 685.64, 687.78, 689.92, 692.06,
694.2, 696.34, 698.49, 700.64, 702.78, 704.92, 707.08, 709.22, 711.37, 713.52, 715.67,
717.82, 719.97, 722.12, 724.28, 726.43, 728.59, 730.74, 732.9, 735.06, 737.22, 739.38,
741.54,  743.7, 745.86, 748.02, 750.18, 752.35, 754.52, 756.68, 758.84, 761.01, 763.18,
765.35, 767.52, 769.69, 771.86, 774.04, 776.2,778.38, 780.56, 782.73, 784.9, 787.08,
789.26, 791.44, 793.62, 795.8, 797.97, 800.16, 802.34, 804.53, 806.71, 808.9, 811.08,
813.27, 815.46, 817.64, 819.84, 822.03, 824.21, 826.4, 828.6, 830.79, 832.98, 835.18,
837.37, 839.57, 841.77, 843.96, 846.16, 848.36, 850.56, 852.76, 854.96, 857.17, 859.37,
861.58, 863.78, 865.99, 868.2, 870.4, 872.61, 874.82, 877.03, 879.24, 881.45, 883.66,
885.88, 888.09, 890.3, 892.52, 894.73, 896.96, 899.18, 901.38, 903.6, 905.82, 908.06,
910.28, 912.5, 914.71, 916.94, 919.16, 921.4, 923.62, 925.85, 928.08, 930.3, 932.54,
934.77,937.0, 939.23, 941.46, 943.7, 945.93, 948.17, 950.4, 952.64, 954.88, 957.12,
959.36, 961.6,963.84, 966.08, 968.32, 970.57, 972.81, 975.06, 977.3, 979.55, 981.8,
984.04, 986.3, 988.54, 990.8, 993.05, 995.3, 997.56, 999.81]

# plot the MTFs for all bands and report MTF50 and area under the curve
fig, ax1 = plt.subplots()
print(np.min(area), np.average(area), np.max(area))
print(np.min(under), np.average(under), np.max(under))
line1, = ax1.plot(wavelength, under, color='red', label = '50%')
ax2 = ax1.twinx()
line2, = ax2.plot(wavelength, area, color='blue', label = 'Area Under MTF Curve')
ax1.set_xlabel('Wavelength (nm)', fontsize=14)
ax1.set_ylabel('Frequency (~/px)', color='r', fontsize=14)
ax2.set_ylabel('Area under MTF Curve (~/px)', color='b', fontsize=14)
```

```
ax1.grid(axis='x')
ax1.legend([line1, line2], ['MTF50', 'Area Under MTF Curve'])
plt.show()
# print(integrate.simpson(freq_50_mtf, band_50_mtf, freq_50_mtf[1] - freq_50_mtf[0]),
#       integrate.simpson(freq_150_mtf, band_150_mtf, freq_150_mtf[1] - freq_150_mtf[0]),
#       integrate.simpson(freq_250_mtf, band_250_mtf, freq_250_mtf[1] - freq_250_mtf[0]))
#
# plt.plot(band_50_mtf, freq_50_mtf, color='red', label = '490 nm')
# plt.plot(band_150_mtf, freq_150_mtf, color='blue', label = '700 nm')
# plt.plot(band_250_mtf, freq_250_mtf, color='green', label = '919 nm')
# plt.plot(band_50_mtf[freq1], freq_50_mtf[freq1], marker = 'o')
# plt.plot(band_150_mtf[freq2], freq_150_mtf[freq2], marker = 'o')
# plt.plot(band_250_mtf[freq3], freq_250_mtf[freq3], marker = 'o')
# plt.xlabel('Spatial Frequency (~/px)', fontsize=16)
# plt.ylabel('Modulation', fontsize=16)
# plt.xticks([band_50_mtf[freq1], band_150_mtf[freq2], band_250_mtf[freq3],
    # 0.5, 0.6, 0.7, 0.8, 0.9, 1])
# plt.ticklabel_format(style = 'scientific', axis = 'x', scilimits=(-2, -2))
# plt.yticks([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
# plt.grid()
# plt.legend()
# plt.show()
```

REFERENCES

[1] Goetz, A., Vane, G., Solomon, J., and Rock, B., "Imaging spectrometry for earth remote sensing," *Science (New York, N.Y.)* **228**, 1147–53 (07 1985).

[2] Lu, B., Dao, P., Liu, J., He, Y., and Shang, J., "Recent advances of hyperspectral imaging technology and applications in agriculture," *Remote Sensing* **12**, 2659 (Aug 2020).

[3] Lu, G. and Fei, B., "Medical hyperspectral imaging: a review," *Journal of biomedical optics* **19**(1), 010901–010901 (2014).

[4] Weisz, E., Smith, N., and Smith Sr, W. L., "The use of hyperspectral sounding information to monitor atmospheric tendencies leading to severe local storms," *Earth and Space Science* **2**(9), 369–377 (2015).

[5] Van der Meer, F. D., Van der Werff, H. M., Van Ruitenbeek, F. J., Hecker, C. A., Bakker, W. H., Noomen, M. F., Van Der Meijde, M., Carranza, E. J. M., De Smeth, J. B., and Woldai, T., "Multi-and hyperspectral geologic remote sensing: A review," *International Journal of Applied Earth Observation and Geoinformation* **14**(1), 112–128 (2012).

[6] Logan, R. D. et al., "Calibration and characterization of a vnir hyperspectral imager for produce monitoring," (2020).

[7] Eismann, M. T., [*Imaging Spectrometer Calibration*], SPIE Press (2012).

[8] Jablonski, J., Durell, C., Slonecker, T., Wong, K., Simon, B., Eichelberger, A., and Osterberg, J., "Best practices in passive remote sensing vnir hyperspectral system hardware calibrations," in [*Hyperspectral Imaging Sensors: Innovative Applications and Sensor Standards 2016*], **9860**, 13–42, SPIE (2016).

[9] "P4001 draft."

[10] Bowen, I., "The image-slicer a device for reducing loss of light at slit of stellar spectrograph.," *The Astrophysical Journal* **88**, 113 (1938).

[11] Weitzel, L., Krabbe, A., Kroker, H., Thatte, N., Tacconi-Garman, L., Cameron, M., and Genzel, R., "3d: The next generation near-infrared imaging spectrometer," *Astronomy and Astrophysics Supplement Series* **119**(3), 531–546 (1996).

[12] Barden, S. C. and Wade, R. A., "Densepak and spectral imaging with fiber optics," in [*Fiber optics in astronomy*], **3**, 113–124 (1988).

[13] Bacon, R., Adam, G., Baranne, A., Courtes, G., Dubet, D., Dubois, J., Emsellem, E., Ferruit, P., Georgelin, Y., Monnet, G., et al., "3d spectrography at high spatial resolution. i. concept and realization of the integral field spectrograph tiger.," *Astronomy and Astrophysics Supplement, v. 113, p. 347* **113**, 347 (1995).

[14] Xiong, Z., Wang, L., Li, H., Liu, D., and Wu, F., "Snapshot hyperspectral light field imaging," in [*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*], 3270–3278 (2017).

[15] Descour, M. R., [*Non-scanning imaging spectrometry*], The University of Arizona (1994).

[16] Gat, N., "Imaging spectroscopy using tunable filters: a review," *Wavelet Applications VII* **4056**, 50–64 (2000).

[17] Oiknine, Y., August, I., Blumberg, D. G., and Stern, A., "Nir hyperspectral compressive imager based on a modified fabry–perot resonator," *Journal of Optics* **20**(4), 044011 (2018).

[18] Elmasry, G., Kamruzzaman, M., Sun, D.-W., and Allen, P., "Principles and applications of hyperspectral imaging in quality evaluation of agro-food products: a review," *Critical reviews in food science and nutrition* **52**(11), 999–1023 (2012).

[19] Inc., R., "User manual for spectronon version 3.4.3," (2022).

[20] Yokoya, N., Miyamura, N., and Iwasaki, A., "Preprocessing of hyperspectral imagery with consideration of smile and keystone properties," in [*Multispectral, Hyperspectral, and Ultraspectral Remote Sensing Technology, Techniques, and Applications III*], **7857**, 73–81, SPIE (2010).

[21] Scientific, H., "Hyperspectral imaging distortions: Keystone and smile." https://static.horiba.com/fileadmin/Horiba/Products/Scientific/OEM/PoliSpectra_M116_MultiTrack_Spectrometer/Hyperspectral_Imaging_Distortions-Keystone_and_Smile.pdf. Accessed: 2023-4-19.

[22] Durell, C., Skauli, T., and Gilchrist, J. R., "An overview of the emerging p4001 hyperspectral standard," (2022).

[23] Labsphere, "Uniformity mapping test report," (2022).

[24] D'Agostino, J. A. and Webb, C. M., "Three-dimensional analysis framework and measurement methodology for imaging system noise," in [*Infrared Imaging Systems: Design, Analysis, Modeling, and Testing II*], **1488**, 110–121, SPIE (1991).

[25] O'Shea, P. and Sousk, S., "Practical issues with 3d noise measurements and application to modern infrared sensors," in [*Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XVI*], **5784**, 262–271, SPIE (2005).

[26] Li, H., Yan, C., and Shao, J., "Measurement of the modulation transfer function of infrared imaging system by modified slant edge method," *Journal of the Optical Society of Korea* **20**(3), 381–388 (2016).

[27] "Introduction to modulation transfer function: Edmund optics," (2023).

[28] Kohm, K., "Modulation transfer function measurement method and results for the orbview-3 high resolution imaging satellite," in [*Proceedings of ISPRS*], **35**, 12–23 (2004).

[29] Haefner, D., "Matlab executables for 3d noise and mtf," (2018).