Spectral Capture at 30fps

May 23, 2012

Author:
Chris Mondiek

Advisor:
David Long

Submitted in partial fulfillment of the requirements for the BS degree in the School of Film and Animation, Rochester Institute of Technology, Rochester, NY

Contents

1 Abstract

Traditional three channel imaging systems are limited in the color accuracy they can produce and also have serious issues with observer metamerism. Spectral imaging involves capturing more than three channels of color information to record all the spectra available at the scene. By recording and then ultimately projecting spectra, observer metamerism can be greatly reduced. Here, a spectral imaging system using six primaries was constructed using two Imaging Source DFK 31BF03 cameras focused through a beamsplitter and Hassleblad lens to a scene. Two Schott filters, VG9 and BG40 were placed over each camera to create the different spectral sensitivities. The images from the two cameras were captured by a PC using C++ for processing and storage. The images were then digitally registered, combined into a spectral image, and stored into an XML file container containing the principle component values which can be used to directly recreate scene spectra for each pixel.

2 Theory / Background

2.1 Trichromatic Theory

2.1.1 Human Cone Response

All traditional color systems available today capture three channels of information, usually red, green, and blue. This system works because humans are also based on trichromacy and have long, medium, and short wavelength cones, which correspond approximately to red, green, and blue spectral sensitivities respectively as shown in the figure below. Each type of cone will generate a signal based on what wavelengths are present and that cone's sensitivity to those wavelengths. This is shown by the below equation where Φ(λ) is the spectral power distribution (SPD) of the stimulus and L(λ), M(λ), S(λ) are the cone spectral responses, and L, M, and S are the generated cone responses.

$$\begin{aligned} L &= \int \phi\left(\lambda\right) L\left(\lambda\right) d\lambda \\ M &= \int \phi\left(\lambda\right) M\left(\lambda\right) d\lambda \\ S &= \int \phi\left(\lambda\right) S\left(\lambda\right) d\lambda \end{aligned}$$

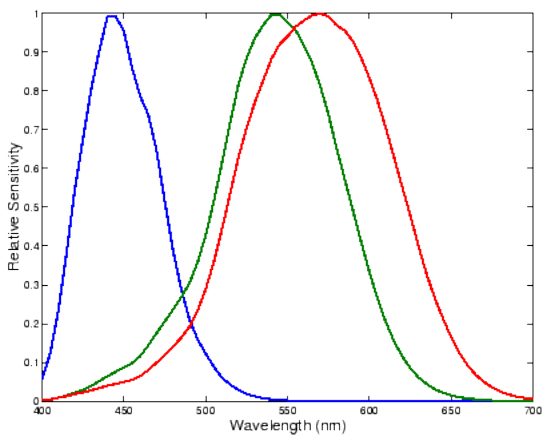


Figure 2.1: LMS sensitivity of eye

2.1.2 Metamerism

Because the generated response for each L M and S is one number based on spectral summation, multiple stimuli SPDs can generate the same response. This idea can be better understood by looking at a simple case of monochromatic sources. A source at ~420nm and 475nm both will generate the same response on the S cone because those two wavelengths both have the same sensitivity for the short cone. This idea can be expanded to a full spectrum source and can be seen in figure 2.2 below. The solid and dotted lines represent two different stimuli. As can be seen from the figure they have quite different SPDs, but they integrate to generate the same LMS responses. These are called metameric pairs and the idea of metamerism is the basis for trichromatic imagery today. Because of metamerism, it is possible to reproduce real world colors without actually reproducing their exact spectra. All that needs to be done is to come up with some stimulus that matches the spectra of the object that is being imaged. The idea can also be used on any system with spectral sensitivities, so cameras, and other devices also are susceptible to metameric pairs. But one pair of metameric matches is only valid for one particular curve, so if that curve is changed, then those two pair might not produce the same color any more.

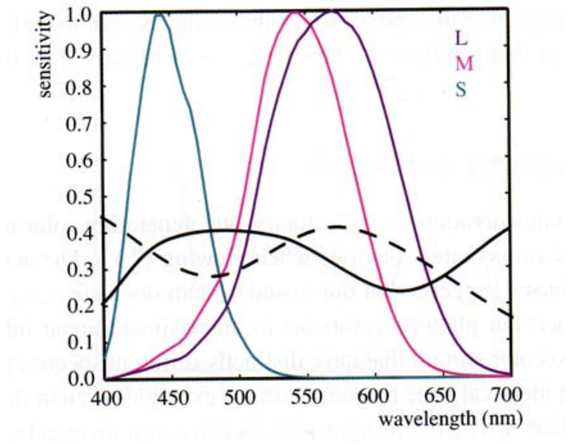


Figure 2.2: Metamerism Example (Image courtesy of Reinhard)

2.1.3 CIE Standard Observer

Experiments were run in 1931 to determine what combinations of three red, green, and blue primaries, at 435.8, 546.1, and 700nm respectively, would match a single monochromatic light. This was done to determine the human color response. The curves shown in the figure below are what was determined by the experiment. It should be noted that the rbar curve goes negative for some values. This is because some colors could not be matched by simply adding the primaries. In some cases negative amounts of some primaries were needed, which was obtained by adding light to the test patch in the experiment.

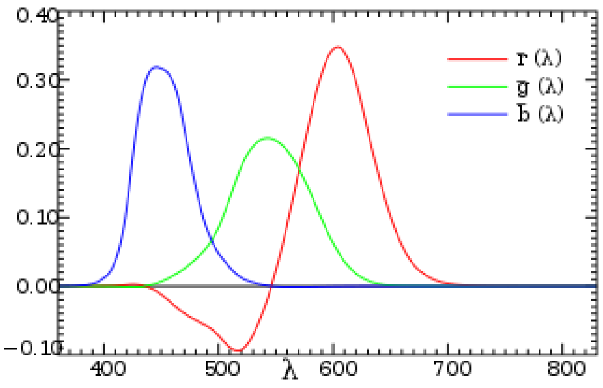


Figure 2.3: rgb color matching functions

It was desired to create a set of all positive color matching functions to ease in calculations and to also integrate the existing photopic response curve V(λ) into the set. This can be done by transforming the rgb color matching functions to a set of color matching functions using imaginary primaries. The use of imaginary primaries is mathematically possible, but not physically realizable, thus it can be used for computations and transformations. After matrixing the RGB values to XYZ tristimulus values one arrives at the curves shown in the figure below. xyz values are simply the responses from the imaginary primaries. XYZ tristimulus values can be found in much the same way as LMS values except using the xyz color matching functions instead of LMS responses as shown below

$$\begin{aligned} X &= \int \phi(\lambda)\, x(\lambda)\, d\lambda \\ Y &= \int \phi(\lambda)\, y(\lambda)\, d\lambda \\ Z &= \int \phi(\lambda)\, z(\lambda)\, d\lambda \end{aligned}$$

XYZ tristimulus are convenient in that they are device independent, meaning that they are based on human cone responses and thus in theory can describe a color independent of any particular device. XYZ values can then be transformed into any device primaries through the use of linear algebra. This is convenient in that it allows for the description of a color throughout the system by serving as an intermediate space.

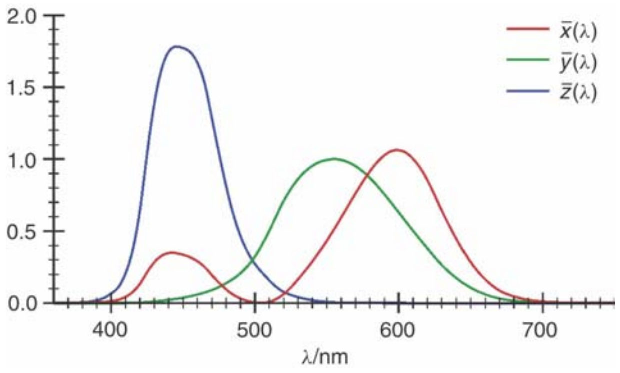


Figure 2.4: xyz color matching functions

2.2 Observer Metamerism

One problem with the using the standard observer as the base for all chromaticity and other calculations is that the standard observer is an average. It is similar to illuminants in that no one actually has the response of the standard observer, because it was based off of an average of the response of 19 people. Every person sees the world slightly differently because everyone has slightly different color matching functions. The standard observer does a good job at approximating everyone's color matching functions and has worked well up to this point. This is an interesting issue in that everyone will see an image presented to them and assume that it is normal. It is only when they talk with other observers about the image that they realize there is a discrepancy.

This is not an issue when viewing a real scene where all spectra is available to the viewer because even if that particular observer's color matching functions are shifted from someone else's, they are still responding to energy at every wavelength as it is present in the real scene. The problem comes when they are viewing projected images from a three channel system, which is showing a metameric match of the original scene but for the standard observer. This has not been a huge issue with imaging systems in the past because of the broad primaries used in projectors and televisions today. The broader the primary the less observer metamerism will be present because even if two peoples color matching functions differ by say a simple wavelength offset, they will both have energy from the source to respond to. With the shift to laser projectors, this becomes more of an issue. Because laser projectors use very narrow primaries, two people could have very different sensitivities to that primary if they have different color matching functions. This is shown in the figure below. The blue curve is the regular ybar function, and the orange curve is ybar shifted by 10nm. As can be seen, the impact with a traditional projector primary is not very severe, but the change in the response from the laser projector is quite pronounced. This project is aimed at reducing observer metamerism.

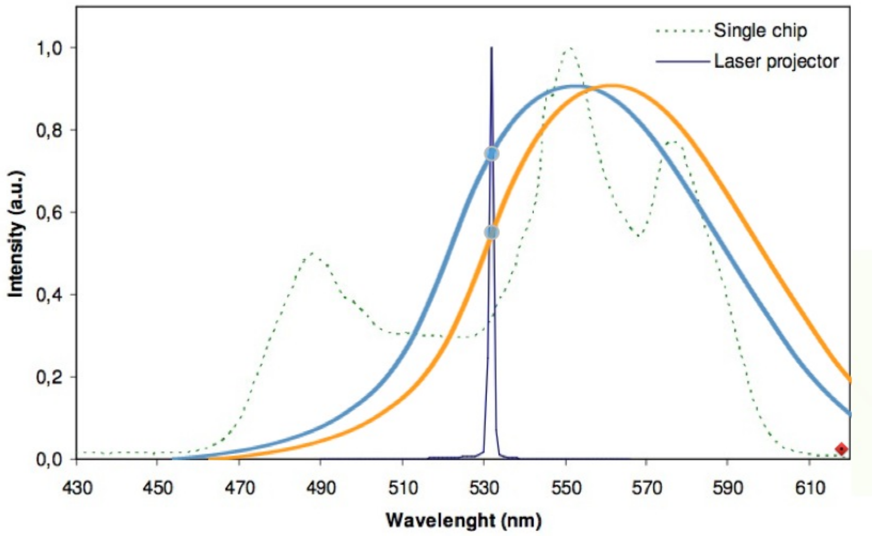


Figure 2.5: Observer Metamerism on projection

2.3 Why Spectral Imaging?

If the cameras capture three channels, the displays show three channels, and humans see with three channels one may ask why bother using more channels for capture or display? Spectral imaging is advantageous for two main reasons.

First it captures more data from the original scene. Traditional three channel cameras sample the available spectra of a scene with only three channels. This means that right from the beginning of the imaging system much of the available information in the scene is thrown away by using only three channels. For every additional channel that the camera records, more information is being taken in from the scene. So in the limit if infinite channels could be taken, one would directly measure scene spectra. This is essentially what a spectrometer does, but the challenge becomes measuring spectra at each pixel location in an image.

A spectral capture system is only physically different from a traditional RGB system in that it has more than three primaries. A full spectral capture system is not feasible in a video application which would involve taking images at 1nm intervals of the same scene. A spectral estimation could be made from three channels of information, but the estimation is greatly improved with the addition of additional channels. By capturing the scene spectra, there is more flexibility when processing the images. Trichromatic systems work by generating a metameric match to the scene spectra using three primaries as mentioned previously, so there is generally some ambiguity to what the color is because what it will ultimately look like is a function of the camera spectral sensitivities, display primaries, display conditions, and observer differences. By switching to spectra as an intermediate space, there is no ambiguity as to what the color should ultimately be because every wavelength is accounted for.

Second, switching to spectral imaging reduces observer metamerism issues. If a multispectral projection system is employed, then observer metamerism becomes less of an issue because an approximation of scene spectra is being projected on the screen. As mentioned before, traditional three channel projectors work by projecting a metameric match to the color that is wanted. The problem comes in that the metameric match is for the standard observer, so if a person does not have the response of the standard observer then the colors they see might be different from what is desired. Reduction of observer metamerism is one of the main objectives of spectral imaging and is analyzed more closely throughout the paper.
It is also important to know how a spectral camera fits into the entire work flow. A spectral camera by itself is relatively useless without a display that can show what is captured by the camera. The diagram below shows the steps involved in getting scene content shown spectrally on a screen. All steps up to and through "conversion to spectral data" are included in this project. Considerations of a spectral display are outside the scope of this project.

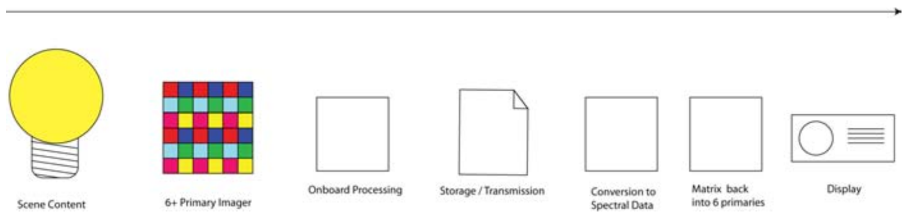


Figure 2.6: Overview of System

3 Design Objectives

There are a variety of ways to build a multispectral camera but for our design there were some parameters that we wanted to achieve. The focus for this project is on reproduction for motion picture and the applications in that field. For that reason a spectral video camera is required that can capture at 24 frames per second (fps) or greater, as is standard for motion picture capture. Raw uncompressed color information is desired as color compression will have less information to work with when doing a spectral estimation. A list of all specifications is given below, with many of them being characteristics that were determined by the end user.

- Raw full color uncompressed 4:4:4 output from camera
  - Need full color because we are trying to reproduce spectra, so uncompressed color is needed
- Minimum of six primaries
- Video frame rates (>24fps)
- HD Resolution
  - 1920x1080 ideal, but we settled for 1280x768 given restrictions on the camera and frame rate
- Ability to change the filters
- Modular
  - Wanted to be able to change parts as the design progressed
- Real time processing
  - Ideally we wanted the spectral estimation to be done as each frame was captured

4 Initial Design Concepts

Because the objective of this camera is to simply capture more than three channels of color information, there are many different ways to approach the design. Below are several initial design concepts for the capture end of the camera.

4.1 Two color filter array cameras through beam splitter

This design option puts two identical color filter array cameras pointing through a white light beam splitter. Different filters are then placed over each camera to yield a total of six different channels. This approach is very feasible and because of that the design that was decided on for actual implementation. This design, while simple to implement is also limited in the ability to create independent color channels. Because each camera already has an RGB color filter array, one is limited to only cutting out sensitivity from those already present.

Pros

- compact / ergonomic
- Simple - very feasible

Cons

- limited channel options as each camera has a fixed RGB CFA

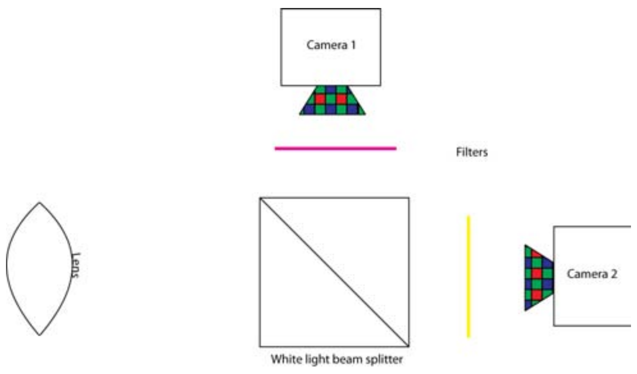


Figure 4.1: Two CFA cameras through a beam splitter

4.2 Three CFA cameras through a X prism

This design uses a X prism as would be found in any modern DLP projector. This option is better than the previous design in that it affords nine channels of information as opposed to six. The problem is that there is less flexibility in the placement of those channels because of the forced splitting of red green and blue light via the X prism, so no channel can be placed across those boundaries.

Pros

- Nine primaries - better spectral estimation

Cons

- requires three cameras
- Spectrum split into distinct red, green, blue segments - Can not have a channel that crosses between the segments.

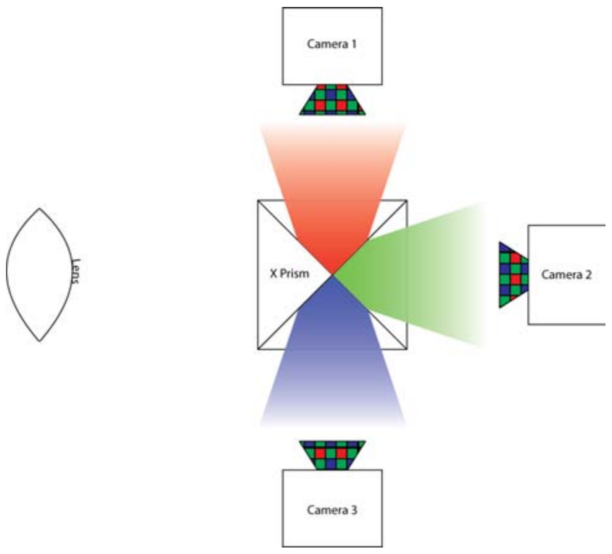


Figure 4.2: Three CFA Cameras through a X Prism

4.3 Six cameras through a X prism & beam splitters

This design improves on the previous one in that it allows slightly more flexibility in channel placement by swapping out the CFA cameras for two monochrome cameras for each red, green, and blue. This then is back to six primaries, but they are full resolution. This option is more complicated than any previous designs and the advantages of full resolution color channels versus the added cost and complexity of the added cameras must be weighed.

Pros

- Flexible

Cons

- Six cameras required

- Very demanding on post processing
- Spectrum split into distinct red, green, blue segments

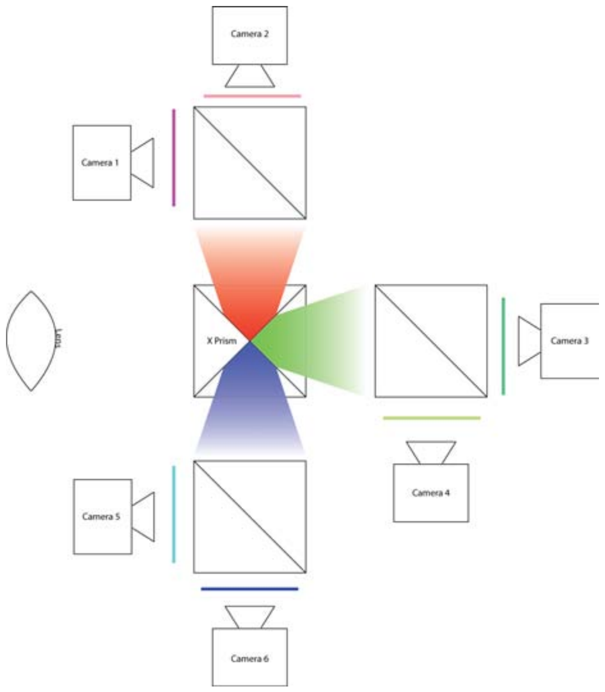


Figure 4.3: Six monochrome cameras through an X prism

4.4 Six monochrome cameras through beam splitters

This is by far the most flexible option of all the designs. it allows for each channel to be completely customized. This however comes at the expense of extreme light loss through a series of beam splitters and thus noise can become a big issue with this system. The design places each camera at locations required to maintain equivalent focal lengths to ensure that the are all in focus. This option also incurs problems with getting six cameras properly aligned so that they are all imaging the exact same scene.

Pros

- Most flexible option
- Completely customizable spectrum
- Full resolution color

Cons

- Low light to each camera - noise issues
- Six cameras required - computationally intensive

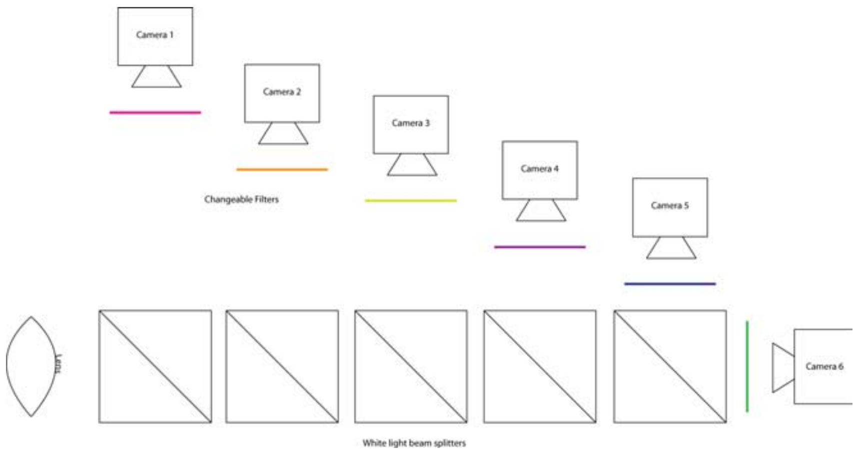


Figure 4.4: Six monochrome cameras through beam splitters

After examining all of the optical configurations described above, option 4.1 “Two color filter array cameras through beam splitter” was chosen as the design to implement for the camera. This design was chosen based on the simplicity of construction. Because of the timeline of this project the other options were not feasible to build in the allotted time frame. Alignment of the cameras as well as signal processing for more than two cameras became impractical. So while some other designs had better performance characteristics, the simplicity of this option outweighed potential performance gains from other configurations.

5 Post Processing Design Options

After the images are captured by the camera presented in the previous section, they must be sent off to a processing block to perform image alignment and spectral estimation before finally being stored in some file format. This is shown in the figure below. Because the goal of this system is to operate in close to real time, doing the full spectral estimation is impractical using most techniques, but one technique using principle component analysis (PCA) can work relatively quickly because it only involves the use of one matrix. Before the spectral estimation is done though, the images need to be aligned. Digital alignment is required because of a lack of awesome hardware to properly align the cameras. If the cameras could be mounted on 5-axis rigs that allowed for meticulous alignment then a digital alignment would not be necessary. This hardware was not available during the construction of the camera and thus a digital method has to be used.

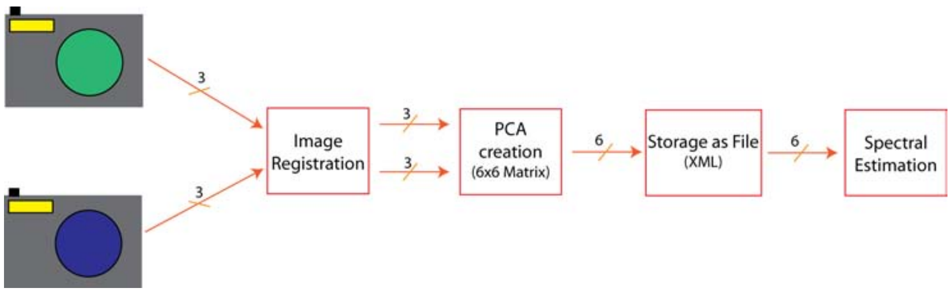


Figure 5.1: Overview of Signal Processing Steps

5.1 Determination of capable hardware

There are several ways to do this post processing, and because the system will deal with six channels of information, equipment that can handle the required data rate is required. The equation below shows how to calculate the required bits per second required by the system to input the data to operate in real time.

$$Resolution \times FrameRate \times \frac{Bits}{Pixel} \times NumberOfChannels = Total\frac{Bits}{Second}$$

For example, a 720p image running at 24fps, with 8bits/pixel for each channel in a six channel system would result in a total of 1.06GB/s. That amount of data needs some very fast hardware to be able to handle that.

$$(1280 \times 720) \times 24 \times 8 \times 6 = 1.06\frac{GB}{s}$$

There are several viable options for handling the data. One option is to simply feed the camera signals into a PC and then do all the post processing in software such as labVIEW or C++. This requires efficient code to allow for a throughput that is close to the desired data rate. Another option is to use a field programmable gate array (FPGA) to do all of the input and processing of the data in one chip. Both options are capable of handling the data and will produce similarly capable systems. Implementation on a PC will allow for more flexibility in the development of the image processing pipeline, while the FPGA solution will result in a more compact easy to use solution.

5.2 Field Programmable Gate Array (FPGA)

An FPGA based solution would handle everything on board a dedicated FPGA chip. The inputs to this system would have to be either GigE cameras or CameraLink cameras with CameraLink being the preferred connection type because those are the only two input types easily connected and dealt with by most FPGA's. The camera signals would be fed into a DS90CR288A chip or similar, which converts to convert the data bits from the camera into data streams that the FPGA can handle. From there the operation would be fairly straightforward with the matrix operation to perform the spectral estimation being hard coded into the FPGA.

The main challenge with this approach is the storage of the data after the spectral estimation. Because the chip would be standalone hardware, external storage would need to be added, and because of the amount of data coming off of the chip the storage would need to be extremely fast. Solutions to this would be to buffer it through sRAM then out to slower media such as hard drives. Other options would be to connect SSDs to the system for storage. Also depending on the exact data rate needed, PCIe or GigE outputs could just be installed.

5.3 PC with software

This solution involves using a PC equipped with the required input cards to capture from the cameras. This card will get the images from the camera and then send them into the computer. Software of some kind is then required to processes the images. Any number of software packages could handle this task. Matlab, C++ running openCV, labVIEW, and a host of others. C++ is the most attractive option due to its efficient utilization of computer resources. This allows for the images to be processed faster and the camera could operate at closer to real time. This option is also incredibly flexible as C++ can be used to implement any algorithm.

One advantage of this method is the ease of adjusting the spectral estimation algorithm. Here it would be relatively straightforward and would involve changing just that part of the code, where as with the FPGA, the entire code would have to be reloaded onto the chip. While an FPGA can be reloaded relatively easily, it requires taking the camera assembly apart, making this option more flexible than an FPGA solution.

5.4 Implementation Choice

The final choice for building the camera was to us a PC implementation. This was done for several reasons. First the cameras that were obtained were the Imaging Source DFK 41BF02 cameras which had firewire 400 outputs. This camera choice eliminated the FPGA option right away. PC implementation was also more flexible and easier to work with and change as the project progressed.

6 Algorithm Details

The following sections go into details of the blocks in the signal processing pipeline from capture out to file storage.

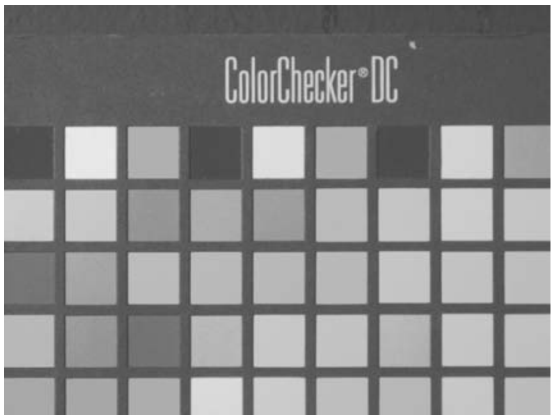
6.1 Image Registration Algorithm

This technique is similar to what is done for digital image stabilization. In image stabilization algorithms, the current image is compared to the previous image in time. In image alignment, the current image is compared to the image from the other camera and moved accordingly. The comparison can be done a variety of ways and various techniques will need to be evaluated before deciding on an option. Edge detection could be done first to accentuate the differences in the image. Local areas chosen at random could also be evaluated to determine which way the image needs to be translated.

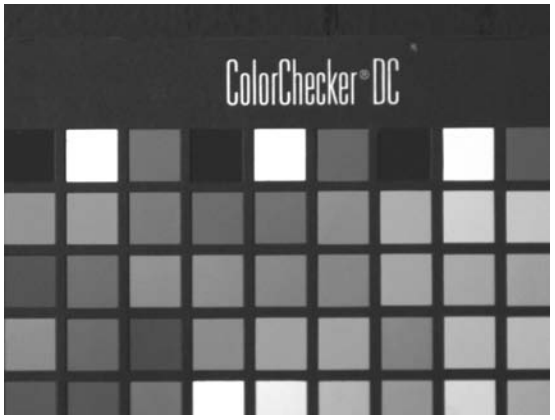
Because digital image registration would not be needed if the cameras could be perfectly aligned, it can be assumed that the amount of change needed will be small. Also to speed up the camera operation, the actual determination of how much to move the image can be done once for the camera and then simply applied to each frame. This option is also preferred in that if the algorithm was trying to determine registration between the two images for every frame, it could mistake motion in the frame for sensor alignment and produce strange results.

Since the algorithm only needs to be run once it can take a very long time to run and produce really good results. ITK toolbox was used to do the image registration. ITK is an opensource library for C++ that has some very good algorithms for doing image registration. The method that was chosen will adjust x and y translation, x and y scale factors, and rotation. The algorithm operates by performing minimization of errors. Basically it will keep one image the same and move the other one to match it. It has a maximum iteration limit of 300 and each time it will adjust parameters and then check and see if the pixels match, it will then change something else and check again. It does this over and over until it has determined that the images match as well as they can. At this point it spits out a text file to be read in by the rest of the program with what changes it made, namely xy translation, xy scale, and rotation.

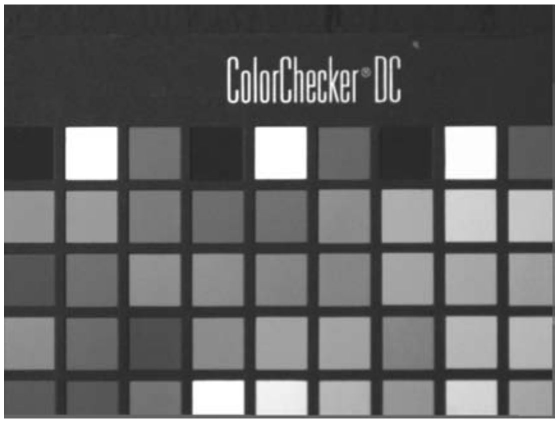
The figure below shows the alignment of two test images. As can be seen, there is not much movement needed because of the care taken in trying to align the images as best as possible. It can also be seen that the moved image has a black border on the bottom and right side. This is one disadvantage of digital image registration. The image must be cropped to accommodate for the movement of the image.



(a) Image kept still



(b) Image to be moved



(c) Result of moving

Figure 6.1: Image registration

6.2 Spectral Estimation Algorithm (Training)

The spectral estimation algorithm consists of two main steps to get to scene spectra: (1) generate principle component values (2) multiply eigen vectors by each principle component value to get an estimation of spectra. This is often referred to as training the camera, basically you are training the camera what to do when it sees different spectra.

6.2.1 Determine Eigen Vectors

These eigen vectors are found from a set of measured spectra. In our case we used a ColorChecker DC target, which has 240 different color patches. Those 240 spectra are then analyzed using principle component analysis. Basically the 31 element vector responsible for the majority of the change in most of the patches is found (31 because of the use of 10nm increments from 400-700nm). A vector responsible for the next most deviation is then found, and so on down to six vectors. It so happens that the first vector is responsible for about 70% of the change, the second for 20%, the third for 5%, etc. So once you get six vectors, one can reproduce those spectra very accurately. The calculated eigen vectors are shown in the figure below. It can be seen that the first vector is essentially an average of the data, then the mode of the vector increases from there, with the second vector being bi-modal, third being tri-modal, and so on.

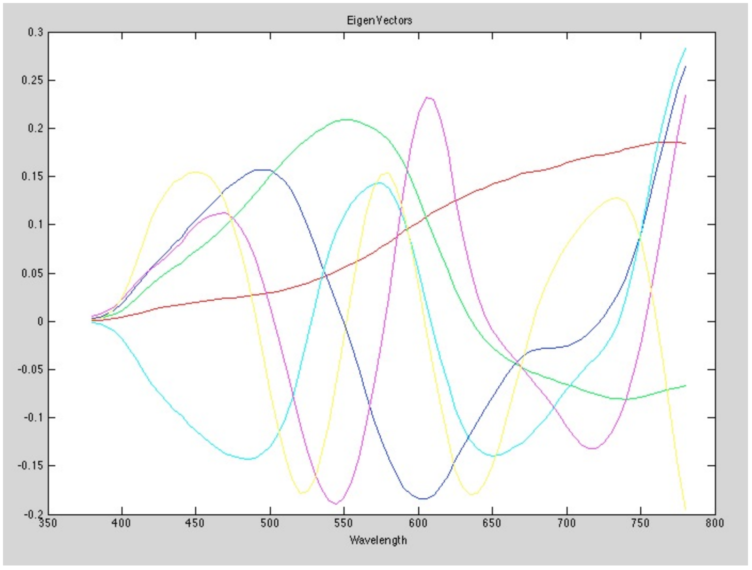


Figure 6.2: Eigen Vectors for the ColorChecker DC

These vectors when properly multiplied will allow for the recreation of any of those 240 measured spectra. It works very well with the worst estimation shown in the figure below, which is still very good.

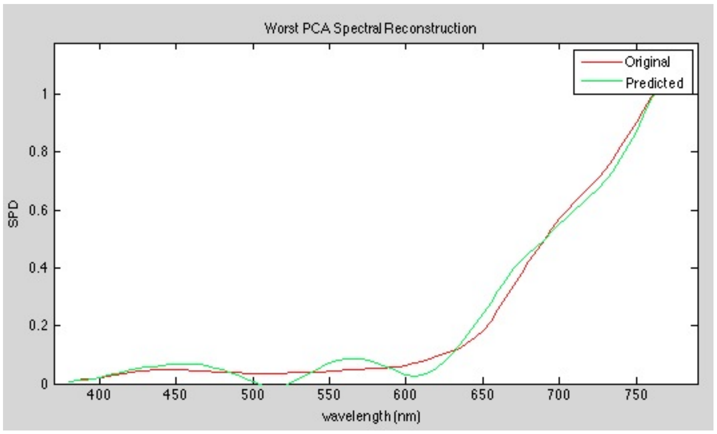


Figure 6.3: Worst PCA recreation of spectra

6.2.2 Determine Principle Component Values

The camera gives out six numbers: R1, G1, B1, R2, G2, B2,as determined by the camera spectral sensitivity. A matrix then needs to be found to go from those camera values to the principle component values. This is a 6x6 matrix because each principle component value is determined from all six camera values. A starting matrix can be found by doing simple matrix math. This matrix is statistically the best matrix to go between the data sets, but it might not be the most visually relevant

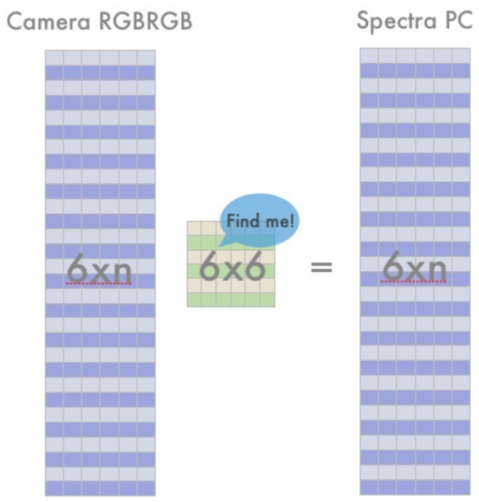


Figure 6.4: Finding the PC matrix

This matrix was optimized to reduce observer metamerism as best as possible. This was done by taking simulated camera RGBRGB signals (simulated with spectral sensitivities + noise) multiplying them by the matrix to get principle component values, then estimating spectra with the eigen vectors. This estimated spectra is then viewed by six simulated observers with different color matching functions. These observers view both the recreated spectra and the actual spectra of the patch and a deltaE between the patches is calculated. This optimization is run until the worst observer's dE is as good as it can be.

6.2.3 Camera Training Considerations

Training the camera is basically the derivation of the matrix done in the previous section. There are a variety of variables involved in the process though: patches use, illuminant used, and filter pair used. Changing any of these options will change how well the spectral estimation works. Basically the camera is only as good as that matrix. As that matrix cannot predict spectra equally well for all patches, some colors suffer. Because of this the choice of training patches is pretty important. For our purposes, a ColorChecker DC target was an easy way to get a lot of colors measured at once. It is not known how well the camera would actually reproduce real world colors though. If spectra available in the real world is not similar to those that the camera is trained on then the spectral prediction will not be very accurate.

The camera will work very well on those exact patches and will then interpolate between what it has been trained on to estimate other spectra. So if the color you want to measure is nothing close to what has been trained then there is a risk that the estimation will not be very good. This is why the illuminant changes the training, because changing the illuminant will change the spectra that the camera sees because the source times the reflectance of the object is what is recorded by the camera.

7 Camera Characterization / Filter Selection

The cameras used in the design need to have some measurements done on them, namely spectral sensitivity, linearity, and noise/speed. Spectral sensitivity is required because the native sensitivity of the camera is needed to (1) determine what filters to choose and (2) ultimately determine the spectral sensitivity of all six channels which is needed to do the spectral estimation. Linearity is required to ensure that the camera is operating in a linear mode, with a gamma of 1.0. This is needed because the spectral estimation algorithm expects linear signals, so if a gamma encoded signal is sent to the algorithm, useless results will be output.

7.1 Linearity

Linearity is tested by providing incrementally greater exposure to the camera and then plotting output code value versus linear exposure values. If the plot is a linear function then the system is linear and it can be assumed that it is operating with a gamma of 1.0, otherwise the camera is not linear. The exposure can be varied by changing either the exposure time or the amount of light entering the lens through the use of neutral density filters. The method chosen was to alter the exposure time while keeping the illumination constant. The image must be of a neutral uniform field. A region of interest in the image that is uniform is then identified and averaged to get an average value for the exposure level. The figure below shows the results for the Imaging Source DFK 31BF03 operating at a setting of gamma = 1.0.

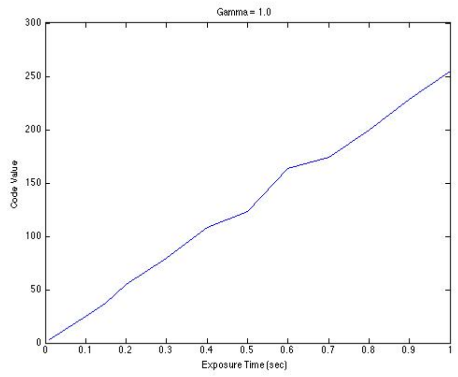


Figure 7.1: Linearity plot at gamma of 1.0

7.2 Spectral Sensitivity

Spectral sensitivity is measured using a monochrometer. A monochrometer is a device that can output light at a specific wavelength using mirrors and gratings. The camera being measured is placed at the exit of the spectrometer and arranged such that the output from the spectrometer is in the center of the image. The spectrometer is then cycled through the visible range starting at 380nm and going out to 720nm in 5nm intervals. The power of the light is also adjusted and measured for each measurement. This is used to normalize the the response of the camera and also ensures that saturation is not reached. The equation for spectral sensitivity in code value per energy is:

$$\frac{CV_{wavelength}}{Radiance_{wavelength}}$$

This calculation is done for every measurement and for each channel. The results are then plotted as CV/Radiance as a function of wavelength. The images below show one image taken by the camera at 555nm and the spectral sensitivity.

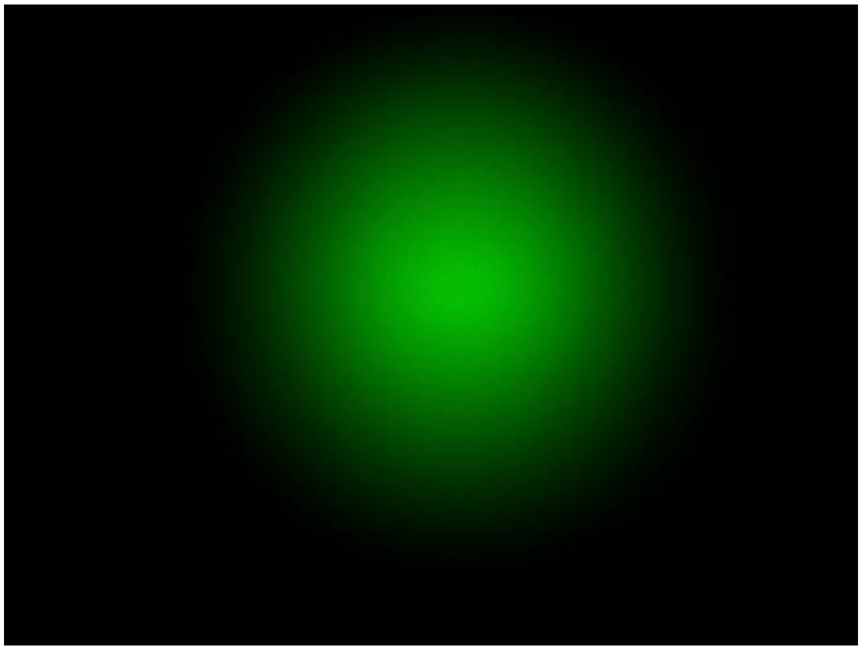


Figure 7.2: Image taken at 555nm

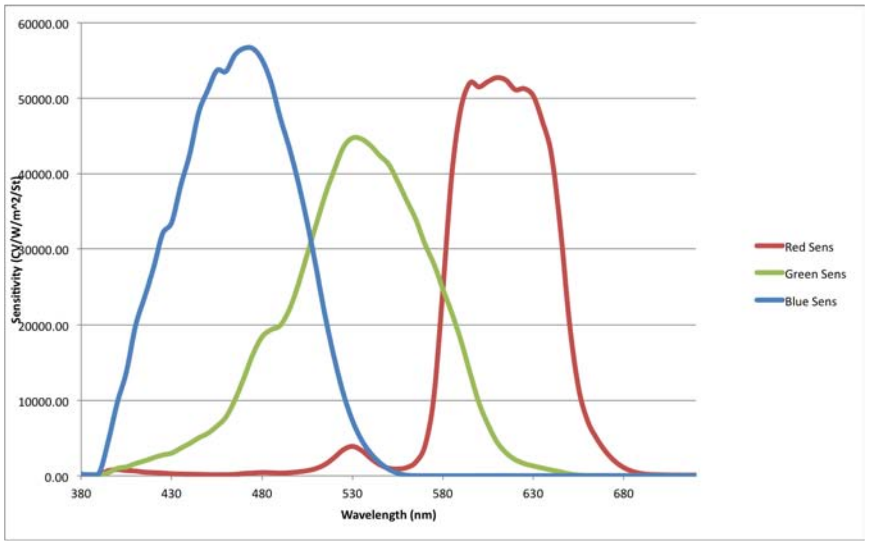
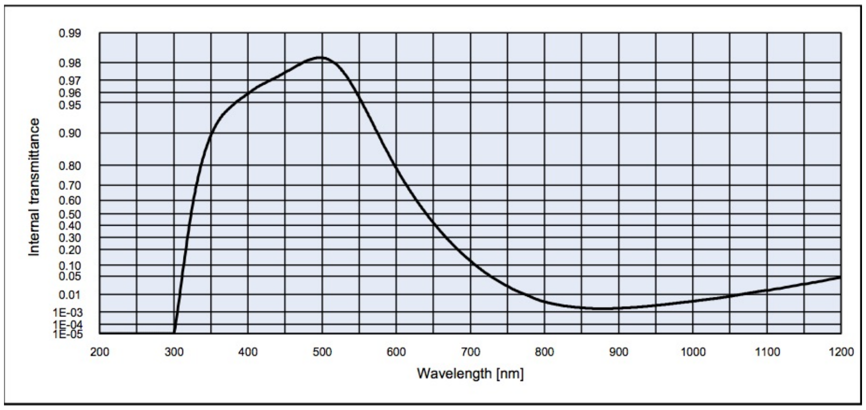


Figure 7.3: Spectral Sensitivity of Imaging Source DFK 31BF03

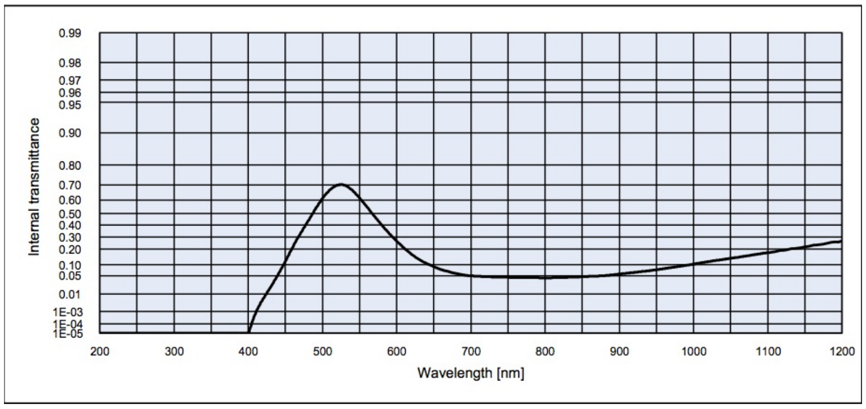
7.2.1 Filter Pair Selection

The filter pair is selected based off of the fact that changing the filter pair will alter the RGBRGB signals output from the camera for each patch. These are then sent through the matrix to convert to principle component values and predict spectra. This spectra is then compared to the original spectra for each patch. various filter pairs are run through this process and it can be determined which filter pair predicts the spectra for all patches the best. It is also the case that different filter pairs might be better for different illuminants. For our purposes, we were shooting a ColorChecker DC target indoors, so we optimized the system for tungsten lighting.

After running this optimization, the filter pair that was selected were Schott Filters, VG9 and BG-40. The transmission profile for these lenses are shown below.



(a) Schott BG-40



(b) Schott VG9

Figure 7.4: Selected Filter Pair Transmissions

Concatenating the filter pairs with the camera spectral sensitivities yields all six channels of spectral sensitivity for the camera.

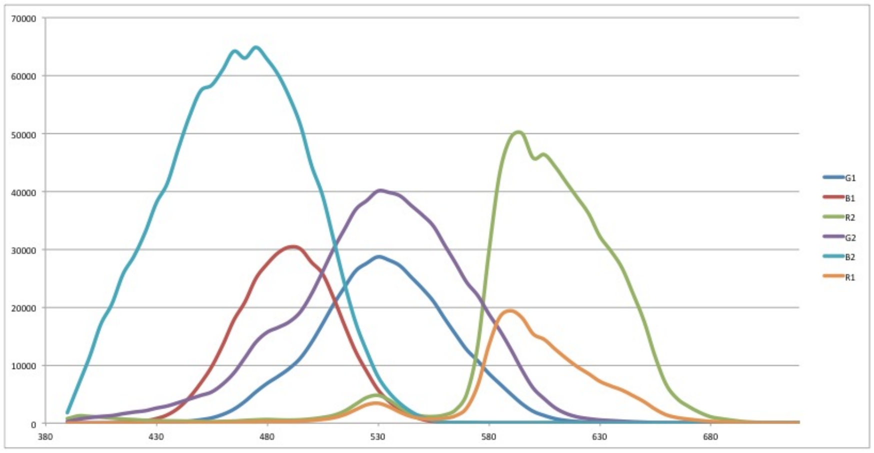


Figure 7.5: Total Camera Spectral Sensitivity

7.3 Verification of Simulations

The filter pair selection and training matrix derivations were both done using a simulated camera. This is done by simply taking the measured camera spectral sensitivities and multiplying them with the scene spectra and the filter pairs. Because so much of the camera was designed via simulations, the accuracy of those simulations had to be verified. This was done by simply taking a picture of the ColorChecker DC target without any filters. The measured RGB values were recorded for every patch on the chart. The simulated camera then “took a picture” of those same patches in its simulation. The error between those two measurements was then calculated. The figure below shows both the absolute and percent errors between the simulations and the actual camera performance for the entire ColorChecker DC chart. The absolute error shows a vertical dependence, which was attributed to lighting non-uniformity. The percent error shows very large spikes of error in the dark patches which is to be expected because of the small magnitude of the patches.

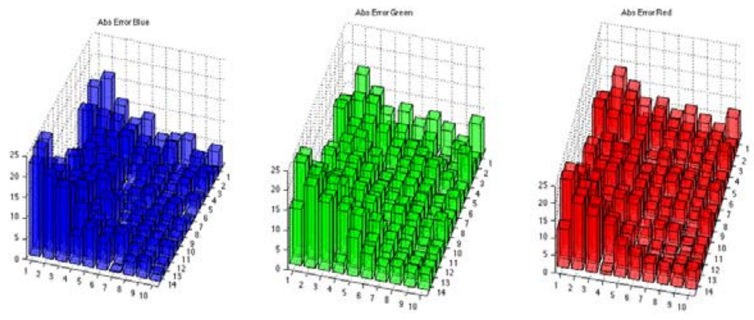


Figure 7.6: Absolute Error

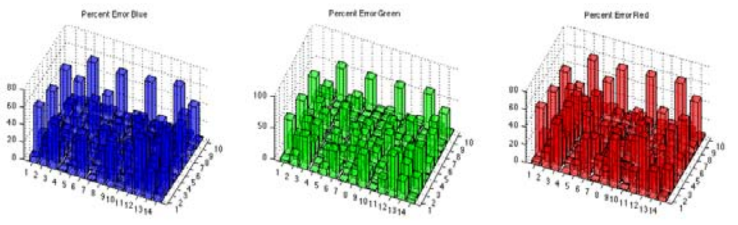


Figure 7.7: Percent Error

What was found by these measurements was the need to simulate a noise floor in the camera. The simulated camera was an ideal camera up to this point and as such did not have any noise. These results showed that in all of the patches there was a small offset. This offset could be attributed to noise in the camera and thus was added to the simulation to better represent the real world camera. The need for the noise floor was especially prevalent in a plot of just gray scale values in the center of the patch. Percent error decreased with increasing luminance for reasons described earlier, but the absolute error remained fairly constant across all luminance levels - a dead giveaway of a noise floor.

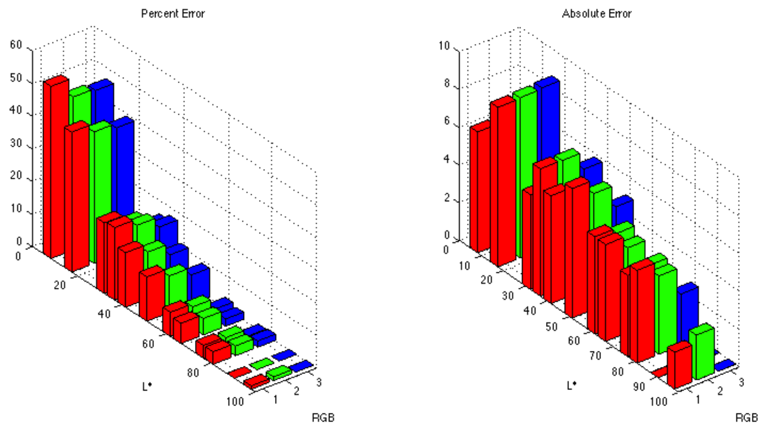


Figure 7.8: Gray scale Error Plots

8 Camera Design Implementation

The general design of the camera changed as the project progressed. Initially it was thought that focusing an image through an objective lens onto each sensor directly would be too difficult to do without extra equipment. Because of this the original design involved focusing the scene onto a diffuser screen and then using another lens on each camera to take a picture of the screen. This method was thought to make alignment of the images easier but was inferior to simply focusing onto the camera sensors in that the diffuser adds grain to the image and also wastes a lot of light making the image on the cameras much darker and noisier. This setup and the images created from it are shown in the figure below.

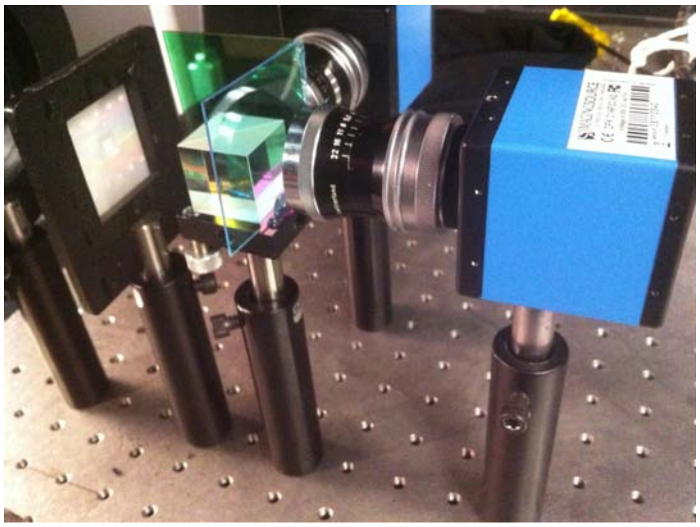


Figure 8.1: Original Camera setup

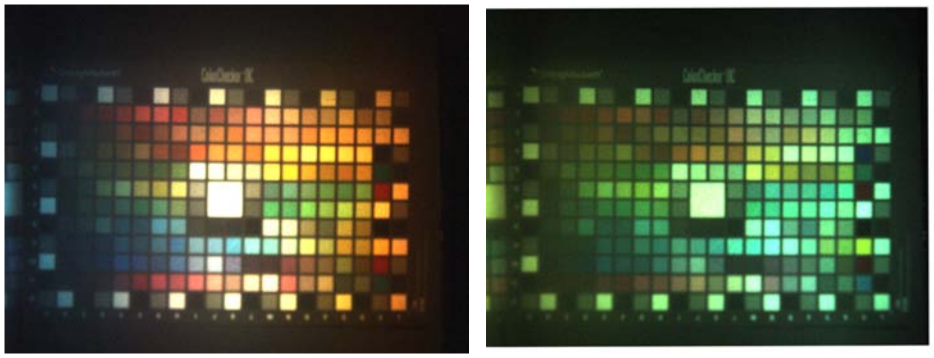

(a) Camera1 (b) Camera2

Figure 8.2: Images from old setup

As can be seen by the images created by this setup, the image quality was very poor. The light loss from the camera required setting the gain to its highest setting making a very noisy image. But more importantly than that, the images had a severe hue shift going from left to right across the image. This obviously made the camera unusable for accurate color measurements. These results were the source of some concern, but it was later determined that the cause of the issue was the combination of the beamsplitter and the lenses used on each camera. The beamsplitter being used was polarizing so it was thought that there was some coating on the lenses being used that caused the hue shift.

Once the lenses were isolated as the problem, we had to try and image without the use of those lenses. Surprisingly we got fantastic results. A Hassleblad lens was used as the objective lens for its increased working distance. The distance from the lens to the imaging plane on a Hassleblad camera is a few inches, which was needed to fit all the components in. The only issue that resulted from doing that was that the image was very zoomed in. a 50mm lens was used but the images that resulted are shown below. Because a Hassleblad is a medium format camera, the imaging plane is 3x3inches, much larger than the 1/3 inch chip in our cameras. All this did was to create a very cropped image, but the images produced by this setup were much superior. The new setup and it's images are shown below.

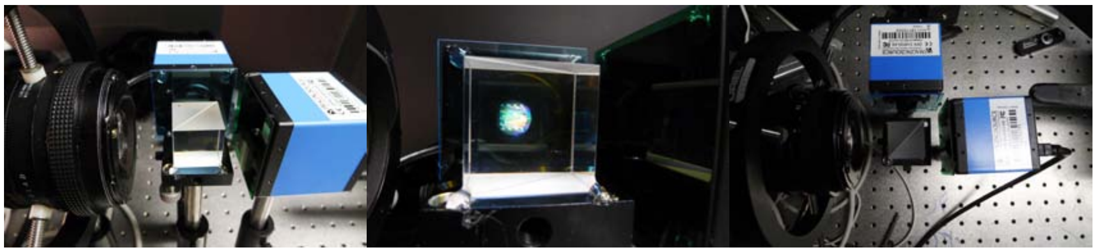
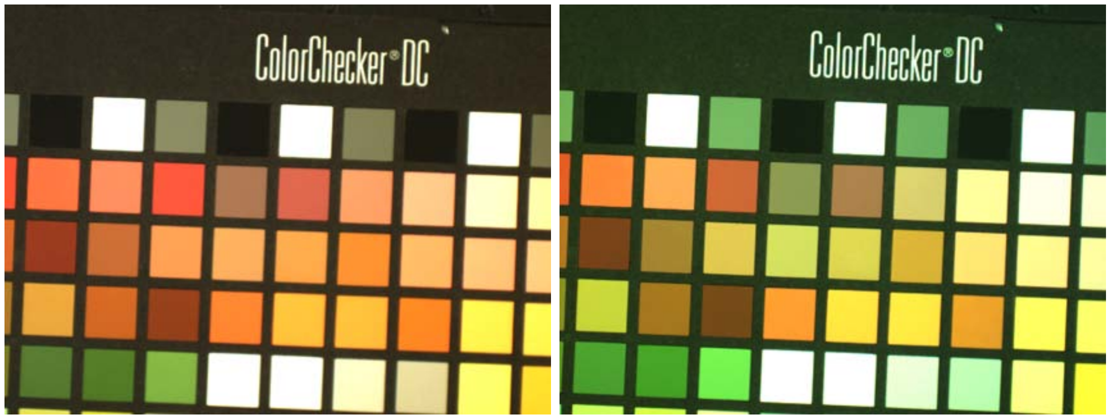


Figure 8.3: New setup



(a) Camera1 (b) Camera2

Figure 8.4: Images from new setup

9 Image Processing Implementation

9.1 How it was done

The implementation of the post processing steps was done using windows and C++. Visual Studio 2010 was used as the development environment. Imaging Source APIs were used to control the cameras and get images into the computer. OpenCV was then used to operate on each video frame and do the spectral estimation. As mentioned earlier ITK was used to do the image registration. The diagram below shows the steps involved in getting a spectral image out of our system. Basically the training matrix is read in from a text file first, and then the registration is run when the camera is first setup. After that one records a video file. Then the processing is done on the already captured video file offline. Processing takes on the order of 10seconds per frame, which is much larger than the .033 which would be required for real time operation, so that metric was not reached.

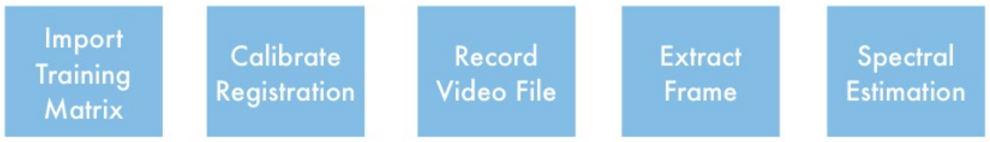


Figure 9.1: Processing Steps

9.2 Saving the Data

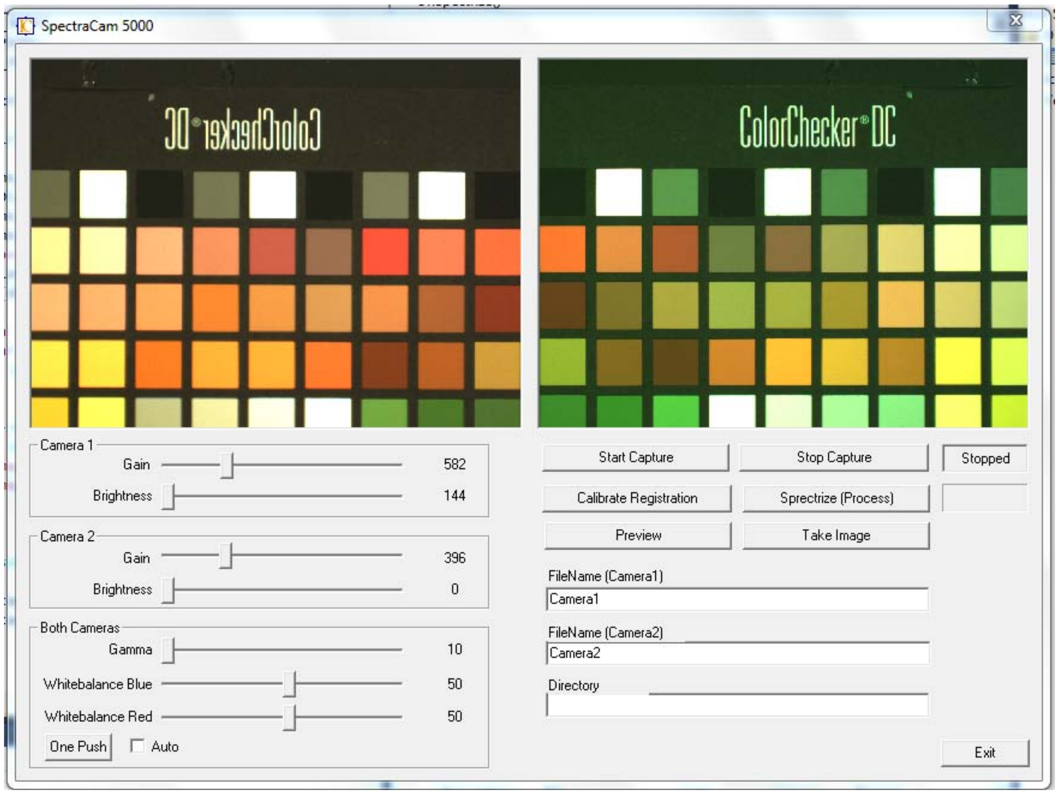
The data that was decided to be stored were the six principle component values. This was done for file size considerations. It would not have been hard to multiply each principle component value by the eigen vectors to recreate spectra for each pixel, but then there is 6x the data to store for each pixel. This makes the file sized unreasonable to work with, so instead the eigen vectors are stored separately and multiplied later. The file size per frame is 81MB, this would jump to 412MB per frame if we stored spectra per frame. 15GB of data per second is simply too big to work with. The data is stored in an XML file which was chosen for its universality. Because the data at that point is not image data, and image container does not make sense, and the PC values are not integers, they are quite complex float values, so some encoding would have to be found. Each XML file stores the size of the frame in its first two tags and then stores the principle components after that. It is stored as a 1D array, in the order

Pixel1						Pixel2						Pixel3					
E1	E2	E3	E4	E5	E6	E1	E2	E3	E4	E5	E6	E1	E2	E3	E4	E5	E6

So the size of the image is used to turn the 1D array into the 2D image array.

10 Using the SpectraCam 5000 Software

Below is a screen capture of the GUI of SpectraCam 5000.



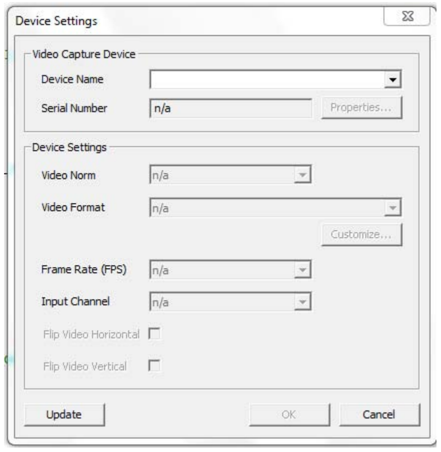
Camera 1
Gain 582
Brightness 144

Camera 2
Gain 396
Brightness 0

Both Cameras
Gamma 10
Whitebalance Blue 50
Whitebalance Red 50

FileName (Camera1) Camera1
FileName (Camera2) Camera2
Directory

Figure 10.1: SpectraCam 5000 Interface


Device Settings

Video Capture Device
Device Name
Serial Number n/a
Properties...

Device Settings
Video Norm n/a
Video Format n/a
Customize...
Frame Rate (FPS) n/a
Input Channel n/a
Flip Video Horizontal
Flip Video Vertical

Update OK Cancel

Figure 10.2: Camera Select Screen

To launch the software currently one must open the solution file in Visual Studio 2010, this can be done by navigating to C:\Users\cam2824\Documents\IC Imaging Control 3.2\samples\vc10\VCD Simple Property and clicking on the VCD Simple Property.sln file. Once the project starts simply click on the green arrow at the top of the program to run the software. This can conceivably be built into a standalone .exe program but that was not realized.

Once the program launches a dialog should come up asking what camera you want to select shown in figure 10.2. There should be two options, choose one and click enter, then the dialog will come up again and choose the other camera. After this you will get the interface shown in figure 10.1. Live view images should be shown of the two cameras automatically, with one of the being backwards due to the beamsplitter. You have options to change the gain and brightness of each camera separately to get the maximum signal possible. The gamma can be changed but should be left at 10, equivalent to a 1.0 gamma. blue and red white balance are tied to both cameras. 50 50 is a good starting value for this. On the right side of the interface is the meat of it. filenames can be specified for each camera. DO NOT put an extension on the name you put in this box. The software will do this automatically depending on what you capture. The directory is the file path to where you want to store the file.

The start capture and stop capture buttons start and stop the capture of a video file respectively. The indicator to the right of the stop capture button will tell if you are currently recording or not. The Calibrate registration button will take the image currently displayed in the live view and analyze it for registration. This will take some time to complete (2-3 minutes). When it finished you will have three images in the location specified by the directory. CalImg1.bmp, CalImg2.bmp, CalOut.bmp. CalImg1.bmp is the image that is not moved, CalImg2.bmp is the image that was moved, and CalOut.bmp is the image with the transformations applied. These images are used to verify that the registration worked properly. The more important part is that it outputs a file named Stats.txt which contains all of the transformations that is applied. This file is what allows the registration to only be run once.

The next button is the Spectrize (Process) Button. This will perform the calculations to get principle component values and store them out to .XML files. Before doing this, the training matrix and offset values must be saved to a .txt file in the order: The top of the figure shows the six offset values and the 6x6 matrix for the order of R1, G1, B1, R2, G2, B2

O_R1	O_G1	O_B1	O_R2	O_G2	O_B1
V1	V2	V3	V4	V5	V6
V7	V8	V9	V10	V11	V12
V13	V14	V15	V16	V17	V18
V19	V20	V21	V22	V23	V24
V25	V26	V27	V28	V29	V30
V31	V32	V33	V34	V35	V36

V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
V29
V30
V31
V32
V33
V34
V35
V36
O_R1
O_G1
O_B1
O_R2
O_G2
O_B2

Before running the Spectrize process, ensure that there is a file called "Spectra" in your specified directory. This is where the files will be stored. One XML file is generated for each frame of the videos specified in the Filename Camera1 and Filename Camera2 boxes.

11 System Performance

So how well does the camera work? After the training matrix was optimized and put into the camera, images were taken and analyzed. The images taken were of the ColorChecker DC target mentioned earlier. The camera was also trained on these patches, so optimal performance is expected for these patches.

11.1 Comparison for all patches

The figures below show RMS error and deltaE 94 errors for all 240 patches. The RMSE plot shows both the spectral estimation errors for the camera as well as the straight PCA error. So the red line in the RMSE plot is as good as the camera will ever do. It can be seen that there are some patches that are higher than we would like, but on the whole it worked pretty well. Some of the deltaE errors are up around 3 which is a bit high but still not very noticeable.

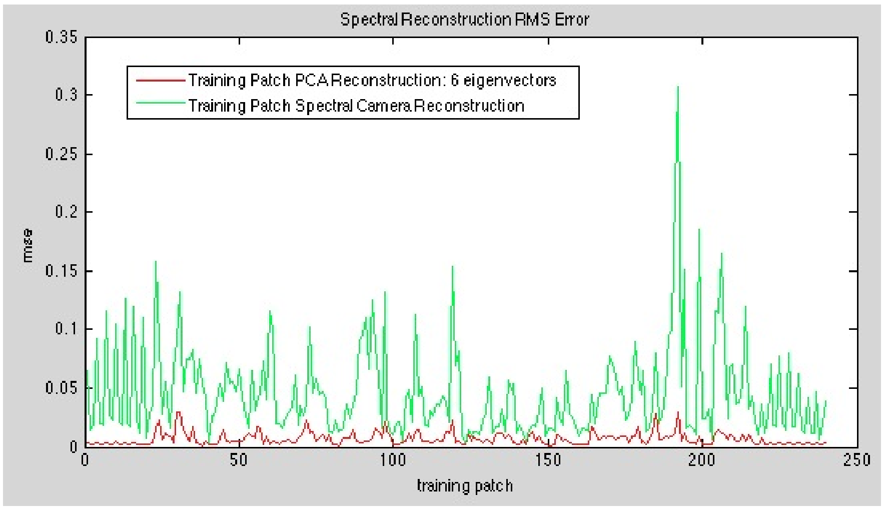


Figure 11.1: RMSE Error Comparison - CCDC Target

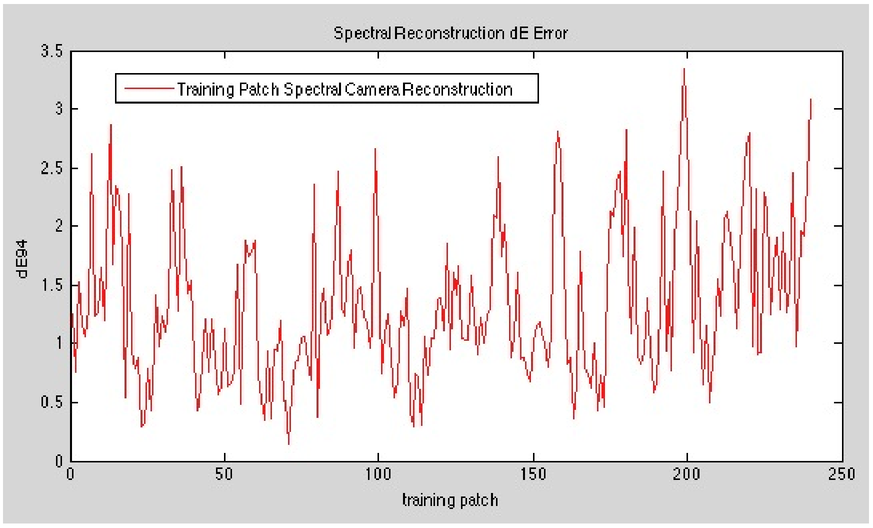


Figure 11.2: DeltaE Error - CCDC Target

11.2 Nine Best/Worst

These plots show the nine best and worst patches for both deltaE and RMSE. It can be seen that these two metrics do not say the same patches are the best. RMSE treats all wavelengths equally in its treatment so it needs all wavelengths from 400 to 700 to match very well with the original spectra. It is also absolute RMSE so it prefers darker patches. DeltaE is weighted by the human sensitivity functions because it is based on XYZ tristimulus values. This means that is places more importance on wavelengths that humans are more sensitive to, namely colors in the green region of the spectrum (500-600nm). This can be seen in the patches that the delta E was the best on. Some of the long wavelengths deviate pretty far, but the colors in the middle wavelengths match very well.
One other thing to note is how when the spectra deviates it is in the longer wavelengths. This is because in those wavelengths the sensitivity of the camera is very low or non existent. This can be seen by checking the spectral sensitivity curves mentioned earlier. But the curves begin to deviate in the high 600nm range. This is about when the sensitivity of the camera is significantly diminished. The camera will not be able to predict spectra well for regions that it is not sensitive. That is one problem with use of filters over cameras, sensitivity can only be taken away, it can't be added.

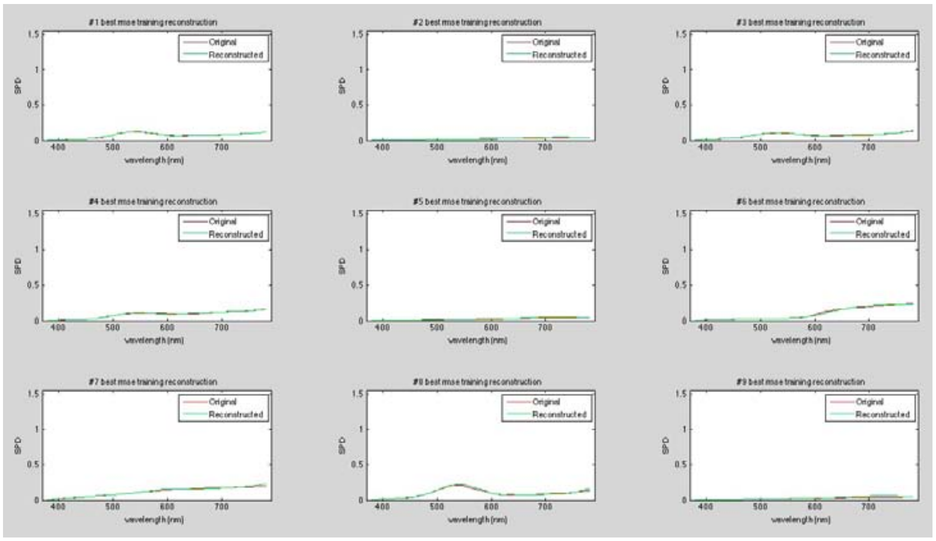


Figure 11.3: RMSE - Nine Best

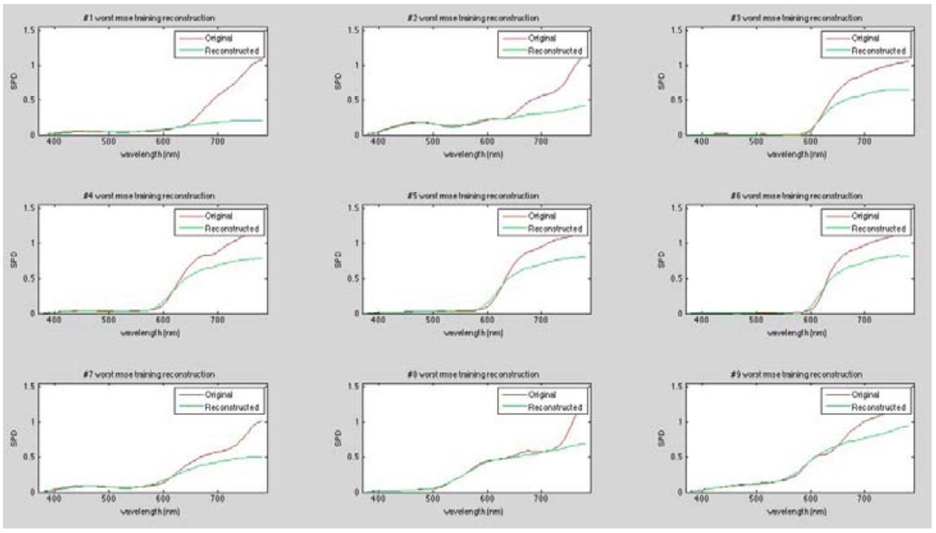


Figure 11.4: RMSE - Nine Worst

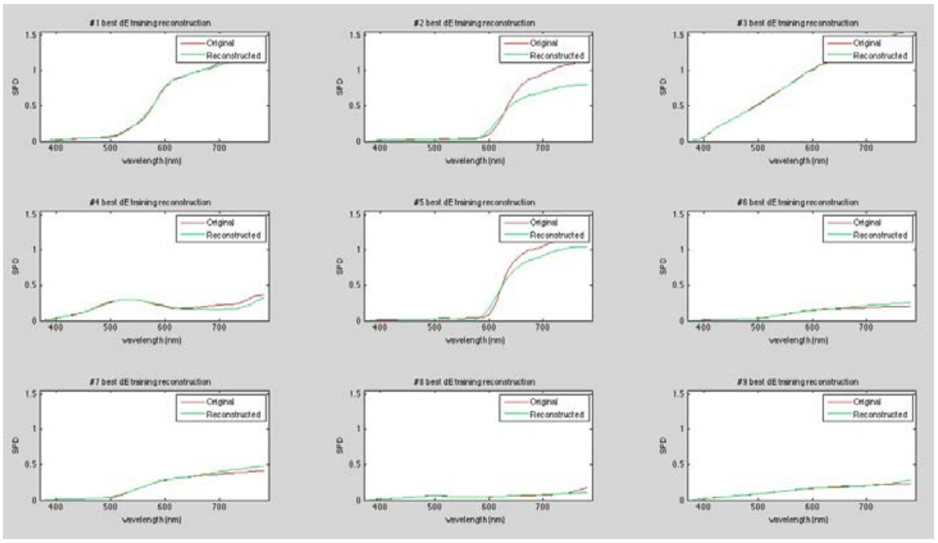


Figure 11.5: deltaE - Nine Best

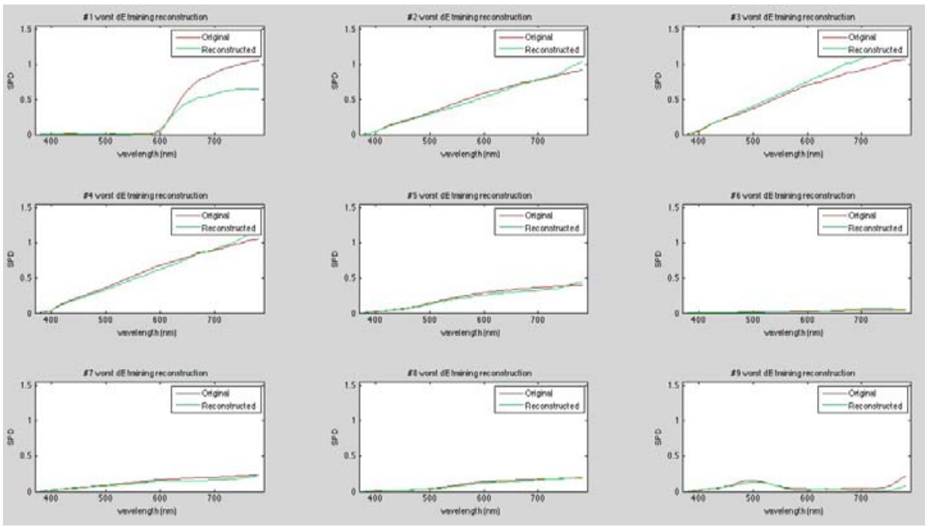


Figure 11.6: deltaE - Nine Worst

11.3 Version 2.0

If version 2.0 of this camera were built, there are several things that would be changed.

The first is to make a more robust casing for the camera. The optical design as of now is quite good and is producing excellent image quality. The issue is that the camera is locked to an optical bench so testing of outside spectra is impossible. The first thing to be done in version 2.0 would be to build an aluminum casing for the camera that allowed it to be moved around and really tested. The actual footprint of the camera is small enough to make to portable (ignoring the required PC computer). Another lens with equivalent working distance but designed for smaller sensors might also be investigated. File format could also be looked at more closely to see if a better file storage solution exists. This would most likely come later when what processing will be done with the signals for output do a display is identified. So for now a flexible XML file format is a good choice to allow for a variety of processing choices and software.

12 Conclusion

In conclusion, the camera was built using two three channel cameras with different filters over each. This generated six total channels of information to work with. Processing was done using a PC and each image was saved to an XML file saving the six principle component values for each pixel. These principle components could then be multiplied by eigen vectors to reproduce scene spectra.

References

[et al.(a)] Masahiro Yamaguchi et al. Color image reproduction based on the multispectral and multiprimary imaging. Technical report, Imaging Science and Engineering Laboratory, a.

[et al.(b)] Massahiro Yamaguchi et al. High-fidelity video and still-image communication based on spectral information. Technical report, Natural Vision, b.

[et al.(c)] R Iwama et al. Real-time multispectral and multiprimary video system. Technical report, Tokyo Institute of Technology, c.

[M. J. vrhel(1992)] H. J. Trussell M. J. vrhel. Color correction using principal components. Technical report, North Carolina State, 1992.

[Reinhard(2008)] Erik Reinhard. *Color Imaging - Fundamentals and Applications.* A K Peters Ltd., 2008.