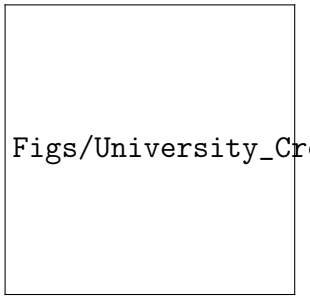
A Fingertip Mechanical-Stress Detector using Chromatic Skin Information



Nayef Al-Saud

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Master of Philosophy in Engineering

Hughes Hall 17 February 2014

I would like to dedicate this thesis to my little one. Let’s never part again.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 15,000 words exclusive of footnotes, appendices and bibliography.

Nayef Al-Saud
17 February 2014

Acknowlegements

I would like to express my deepest gratitude to my supervisor, Prof. Roberto Cipolla, and my advisor, Dr. Joan Lasenby, for their guidance over the past two years, as well as their patience and understanding as I was learning the metaphorical computer vision ropes. I would also like to acknowledge my colleague and friend Dr. Vijay Badrinarayanan, who was always around to offer advice, suggestions or simply a kind word whenever I was working too hard. Most importantly, I woud like to thank my wonderful family and friends in Saudi, the states and elsewhere who supported me through the most difficult period of my life.

Abstract

Human skin and feature detection are common and widespread applications in computer vision. However, many of the algorithms used by such applications are very resource-intensive and require powerful machines to run them with any reasonable level of performance, if at all. Furthermore, these algorithms tend to focus on detection as opposed to retention of information. The purpose of this project is to design and implement a color space algorithm which not only quickly and accurately detects chromatic skin and related features, but is also efficient enough to run on CCD camera-equipped mobile devices. In the first chapter, we examine techniques and color spaces used in typical skin detection algorithms, exploring the benefits and costs of each, as well as physiological considerations and how CCD devices capture color information, and make the case for a bespoke color space transform. The second chapter describes, in exhaustive detail, the design and construction of the bespoke color space, which stores the chromatic and luminosity information separately; retains color space information in a targeted way, discarding irrelevant data based on a given statistical skin model; and performs the color space transform using integer types, thereby eliminating the need for costly floating-point operations. Chapter three outlines the method of gathering the chromatic skin statistics used to build the bespoke color space described in the second chapter, which is expressed as a 2D Gaussian in the chromatic plane, obtained as a product of two 1D Gaussians, thereby further reducing the data processing cost of the algorithm. The fourth chapter describes the practical application of the bespoke color space based on the theoretical designs in the previous two chapters. In the case of this project, it is a mechanical fingertip-stress detector built using the OpenCV computer vision library in C++ and running on an Apple iPhone 6 Plus with iOS version 10.3.3. In the fifth chapter, we evaluate the results of the application by comparing it with a more typical floating-point implementation, which shows a significant four-to-fivefold improvement over the standard approach.

Contents

List of Figures

Chapter 1

Motivation

1.1 Introduction

The purpose of this project is to create a chromatic skin and feature detector for application in mobile devices. Using a given device's built-in CCD camera, objects with characteristics matching those of human skin are identified. This presents a number of challenges. Since we are trying to use the chromatic information of human skin to distinguish objects (e.g. hands, fingers) in a given scene, working in a chromatic space helps to simplify this problem. A chromatic space is a color space wherein the color information is laid out in as few dimensions as possible, with a separate dimension for luminosity — the brightness information. Fundamentally, the issue with this is that CCD cameras capture information using RGB values which mixes the chromatic information with the luminosity. Separating this information is the first part of the challenge.

The second part of the challenge is, because we are searching for objects which exhibit certain chromatic characteristics, a statistical model must be developed and applied in order to characterize the objects as such.

The third and final challenge is developing efficient, discrete maths to perform the chromatic space rotation and to apply the statistical model efficiently enough such that all the chromatic information about the target object is retained, whilst all irrelevant information is discarded.

While several people have developed algorithms for skin detection, their focus has been squarely on detection rather than retention of information. For color spaces, Hue-based spaces, such as HSV, have been used due to the clear separation of the chromatic information and luminosity (**??**). Simpler color spaces, such as Normalized RGB, have been used in video applications due to the demand of continuously processing image frames (**?**). As for

gathering statistics, histogram thresholding (**??**) and Gaussian models using 2D Gaussians (**?**), 3D Gaussians or multiple Gaussian clusters (**?**) are among the most common models in practice (**?**), though other models have been used to similar effect, such as the Self-Organizing Map (SOM) in **?**'s application. Additionally, practically every application uses double precision numbers in their transformations (**???**).

Regardless of the color spaces and statistical models used, all of these algorithms have the same fundamental approach: the image is transformed into some color space, and the statistical model is applied, resulting in a binary image which classifies whether any given pixel is skin. This information is then used as a mask and applied to the original image. This algorithm is fundamentally different to the one used for this project; the primary goal is to preserve information about the skin, as opposed to simply detecting whether a given object is skin or not. We process the image into a new chromatic space, then apply the statistical model, resulting in an image which contains all the chromatic and luminosity information about the skin within it, and information about non-skin areas is lost.

We are preserving the information in a targeted way, reducing the overall information in the image whilst preserving the relevant information about the object we're interested in. This is a clear difference from the more common binary categorization approach, though it is possible to adapt our approach to the same process. However, as it stands, the entire process is more efficient than the first step in a binary classifier, and should be faster than moving to an HSV (Hue, Saturation and Value) image using standard library routines.

1.2 Choosing a Color Space

As mentioned previously, one of the challenges facing this project is identifying a chromatic space in which the color information and luminosity from the raw RGB image can be separated, thereby simplifying the process of identification of human skin color. To this end, the most widely-used chromatic spaces in practical applications of skin color classification have been evaluated — HSV, LAB and YCbCr — all of which have a number of readily-available implementations (**???????**).

Unlike RGB, the HSV color space makes a clear distinction between the chrominance (the "Hue" and "Saturation" channels) and the luminance (the "Value" channel), storing them separately (**??**). However, the chromatic information in the Hue channel is expressed in polar coordinates, thereby necessitating a coordinate transform when converting from the raw image data. Given the nature of the algorithm outlined herein — targeted preservation of information, as opposed to classification — the computational cost of this transformation

is undesirable.

LAB and YCbCr, on the other hand, explicitly separate the chrominance and luminance without an accompanying polar coordinate transform (**???**). LAB and YCbCr use a matrix transform to convert from RGB, allowing for simple and quick conversion exploiting optimized matrix multiplication routines. However, while they appear to be perfectly suited for the purposes of this project, a number of issues arise when represented discretely. LAB, YCbCr and other such luminosity-chrominance spaces all include implicit white-point correction in the readily-available implementations, which distorts the information from the raw image. Also, while the transform is reversible mathematically, discrete data types are used in practical computer science applications, and in most implementations the output is the same data type as the input, resulting in some loss of information when the transform is applied.

Furthermore, based on the skin statistics we have gathered — which will be described in detail in the next chapter — the entire region of skin color in the chromatic space can be expressed as a 2D Gaussian. Typically, this is a complex operation, but by aligning the chromatic axes of a luminosity-chrominance space with the major and minor axes of the 2D Gaussian, it can be expressed as the product of two 1D Gaussians in each chromatic channel, thereby facilitating the application of the statistics. It was decided that a bespoke color space would be best suited for this project.

This new bespoke color space transformation will fulfil the following design criteria:

- Be expressible as a matrix transformation of the RGB values.
- Allow an arbitrary orientation of the chromatic axis.
- Be expressed as an integer type at every step of the transformation, eliminating floating point operations and allowing for efficient optimization on mobile processors.
- Allow for retention and elimination of color space information, to be performed in a non-linear fashion using a statistical specification of the redistribution based on a normal distribution.

1.3 Physiology Study

Now that we have chosen our chromatic space, we will look at the necessary biological considerations. Specifically, the chromophores in human skin and how they affect how skin appears to different detection devices. In brief, chromophores are chemicals in the skin

which interact with light at visible wavelengths, i.e. chromophores are the chemicals which are responsible for the color of skin (**?**).

The aim of this physiology study is to determine, theoretically, how various sensors perceive human skin. Empirically, we have the absorption spectra for various key chromophores, and we have a response spectra for the sensors. We wish to combine this empirical data to predict the output from the sensor.

1.3.1 Chromophores in Human Skin

In the case of human skin, the chemicals responsible for both scattering and absorption are different; the absorption chemicals affect the hue, while the scattering chemicals determine the average path length of the light through the skin, thereby determining the saturation — the amount of light which is absorbed. The longer the path length, the more light is absorbed for any given concentration of a chromophore.

While there are a variety of chromophores, we will be focusing on the four which most significantly affect skin color: oxy-hemoglobin, deoxy-hemoglobin, eumelanin and pheomelanin. The absorption spectra for these chromophores are known quantities, which we can use to find the absorbance of skin, "absorbance" in this case being the ratio of remitted light to absorbed light. The absorbance can be found by applying Beer-Lambert Law (**?**):

$$A = \log \frac{I_0}{I} = \varepsilon l c \tag{1.1}$$

Where A is the absorbance, I_0 is the intensity of the light passing through the reference cell, I is the intensity of the light passing through the sample cell, l is the path length of light through the skin, and c is the concentration of the chromophore. ε is variously called "molar absorptivity," the "molar extinction coefficient," and the "molar absorption coefficient;" all equivalent terms for how strongly a chemical species absorbs light at a given wavelength.

In order to apply the molar absorptivity, we need the average path length of skin, the average concentration of the chromophore, and their Beer-Lambert Law relation — in this case, the product of them all. Unfortunately, the units are not consistent (**?**), but they can each be converted into the SI units, m^2/mol. The data on the chromophores were obtained from the OMLC website, which contains information on the absorption spectra for many different chemicals, including the aforementioned four skin chromophores (**?**).

So, our first task is to convert everything into consistent SI units, and then find the absorbance of the skin using the Beer-Lambert Law relation. As the average path length is a

property of the skin, it is useful to calculate an absorption coefficient for each chromophore; this differs from the molar absorptivity only in that it contains the concentration for the chromophore. We're interested in average values, and the actual concentrations vary markedly between individuals. So we chose a molar concentration equivalent to that of hemoglobin in blood for each of the chromophores, and used their individual molar masses.

The absorption coefficient $\mu = \frac{\varepsilon\rho}{M_{chem}}$, where ρ is the mass concentration and M_{chem} is the molar mass of the chromophore.

We can convert the molar absorptivity into an absorption spectrum for a chromophore if we know the average path length in the skin. So, the scattering properties of the skin will determine the path length. Since we are not trying to simulate any skin in particular, we can simply use average values taken from the OMLC site (**?**). At this point, we aren't so much interested in the saturation — the concentration and path length both affect the saturation of the color rather than the hue — as we are interested in the relative position around the luminosity axis.

So now we have an absorption spectrum, but the color that we perceive is the light which is remitted, and that is what we are really interested in: the remitted light, or R. While the optics is a complicated study, there is a relatively simple theory called Kubelka-Munk Theory (**???**), which allows us to turn an absorption spectra into a remission spectra:

$$r = \frac{A}{S} \tag{1.2}$$

$$r = \frac{(R-1)^2 - T^2}{2R} \tag{1.3}$$

$$S = \frac{\tanh^{-1}\left(\frac{\sqrt{r(r+2)}R}{1-(r+1)R}\right)}{d\sqrt{r(r+2)}} \tag{1.4}$$

Where A is the absorption, S is the scattering of light, R is the remission, and T is transmittance. As the transmittance T of human skin is approximately zero at visible wavelengths, $\frac{A}{S}$ can be expressed as:

$$\frac{A}{S} = \frac{(R-1)^2}{2R} \quad \therefore \tag{1.5}$$

$$R = r + 1 \pm \sqrt{r^2 + 2r} \tag{1.6}$$

As r is $\frac{A}{S}$, the remission R must go down with increasing absorption A:

$$R = r + 1 - \sqrt{r^2 + 2r} \tag{1.7}$$

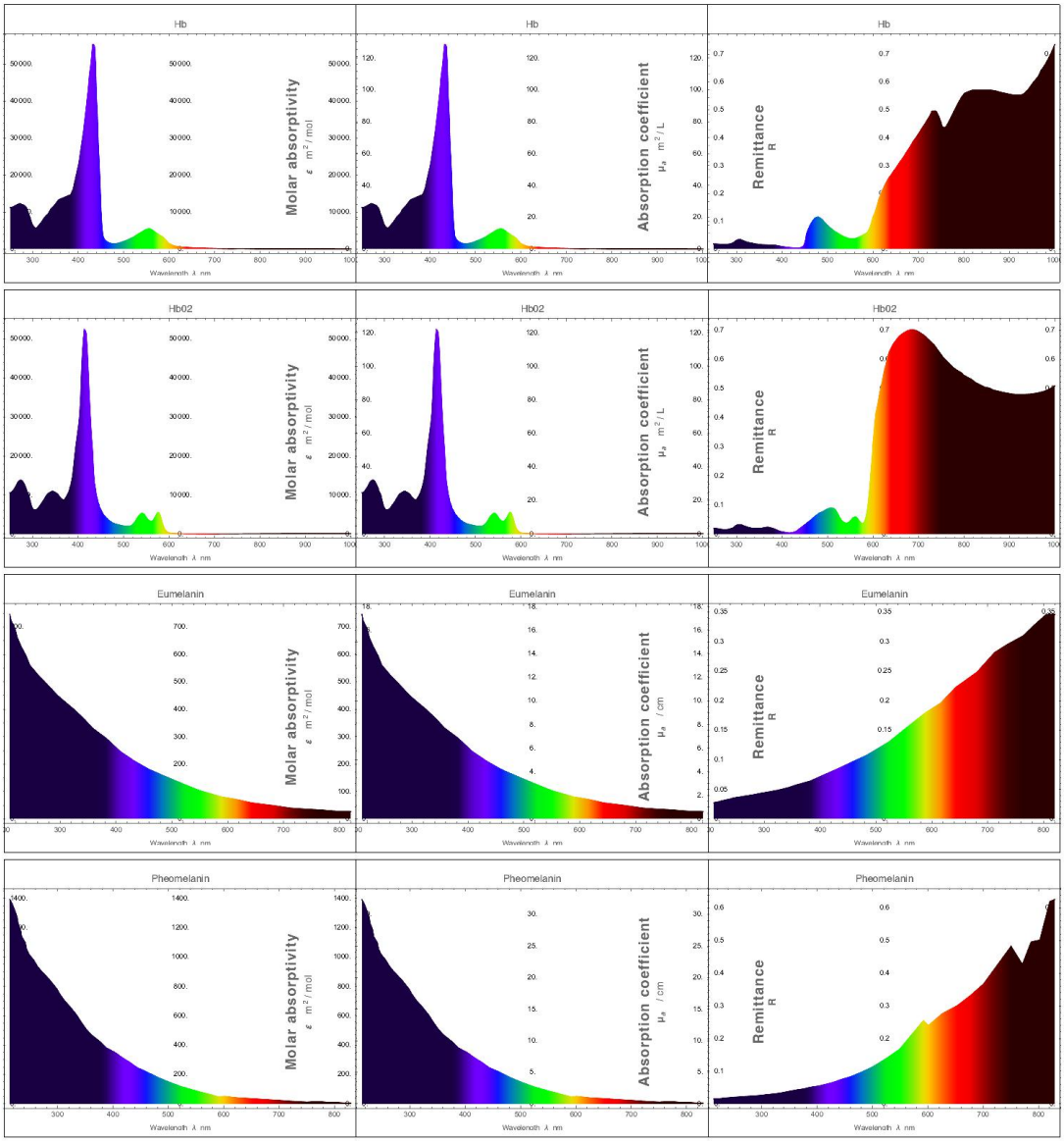


Fig. 1.1 From left to right, the molar absorptivity, the absorption coefficient, and remittance of the four key chromophores. We assume a typical hemoglobin concentration of 150 g/L. The remittance was found using Kubelka-Munk Theory, outlined in section 1.3.3.

The remittance spectra for the four key chromophores is shown in Figure 1.1.

1.3.2 Response Spectra

The response spectrum is how a given sensor — such as the human eye or a CCD — responds to very specific wavelengths of light. Essentially, it is the raw output from the sensor. This is almost never used, and is generally inaccessible even at the hardware layer, as the circuitry very close to the CCD does a basic level of color rebalancing. As we don't know the full color correction for every possible CCD, we observed the raw output spectra of several different devices and then tried two very simple methods of rebalancing the color:

The first is peak normalization, which simply levels out the maximum output in each channel such that the peak output for each of the channels is equal. This approach is very simple, and is quite likely from an electrical engineering perspective as it facilitates the digitization of the color information.

The second method is to approach the response spectra as the basis functions for representing the spectrum. Using this approach, each basis function is normalized according to its integral over the visible wavelengths, giving an equal weighting to each of the response spectra functions over the range of values, rather than focusing on a peak value. Mathematically speaking, this approach is not unusual when constructing a basis set as it facilitates the maths.

It should be noted that the color balancing methods used in modern CCD devices are much more sophisticated than either of these approaches.

However, because the methods chosen here are straightforward amplifications of each channel, they're likely to be used as a first step in color rebalancing at the hardware level in the device. While we are not concerned with device constructions, it is useful to know how these devices capture the color information if this algorithm is to be used diagnostically, as color rebalancing may introduce artefacts and distortions in the data.

Although we consider these sensors to be capturing information throughout the entire visible spectrum, what they are actually capturing varies quite significantly between devices, as shown in Figure 1.3. Even a very simple color rebalancing can restore some normality to the spectrum, as shown in Figure 1.4. However, as these devices use the aforementioned color rebalancing algorithms to alter the image to look more sensible to human eyes, these differences aren't so apparent to us.

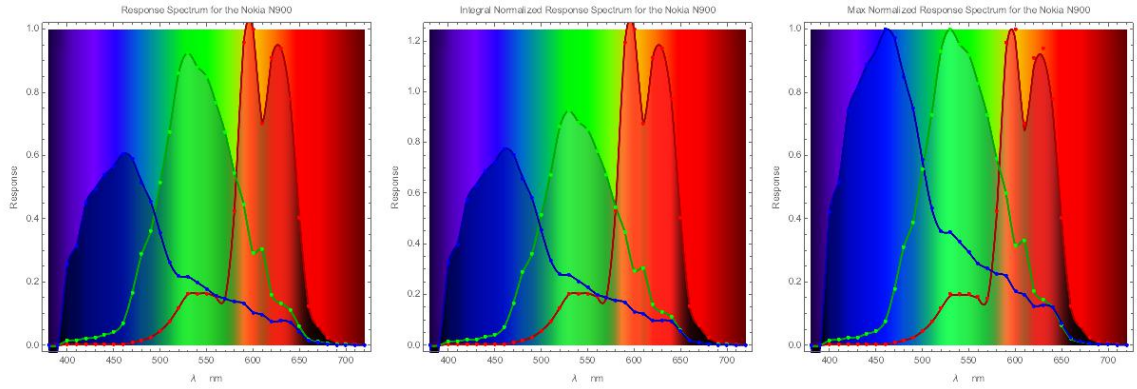


Fig. 1.2 The response spectrum for the Nokia N900.

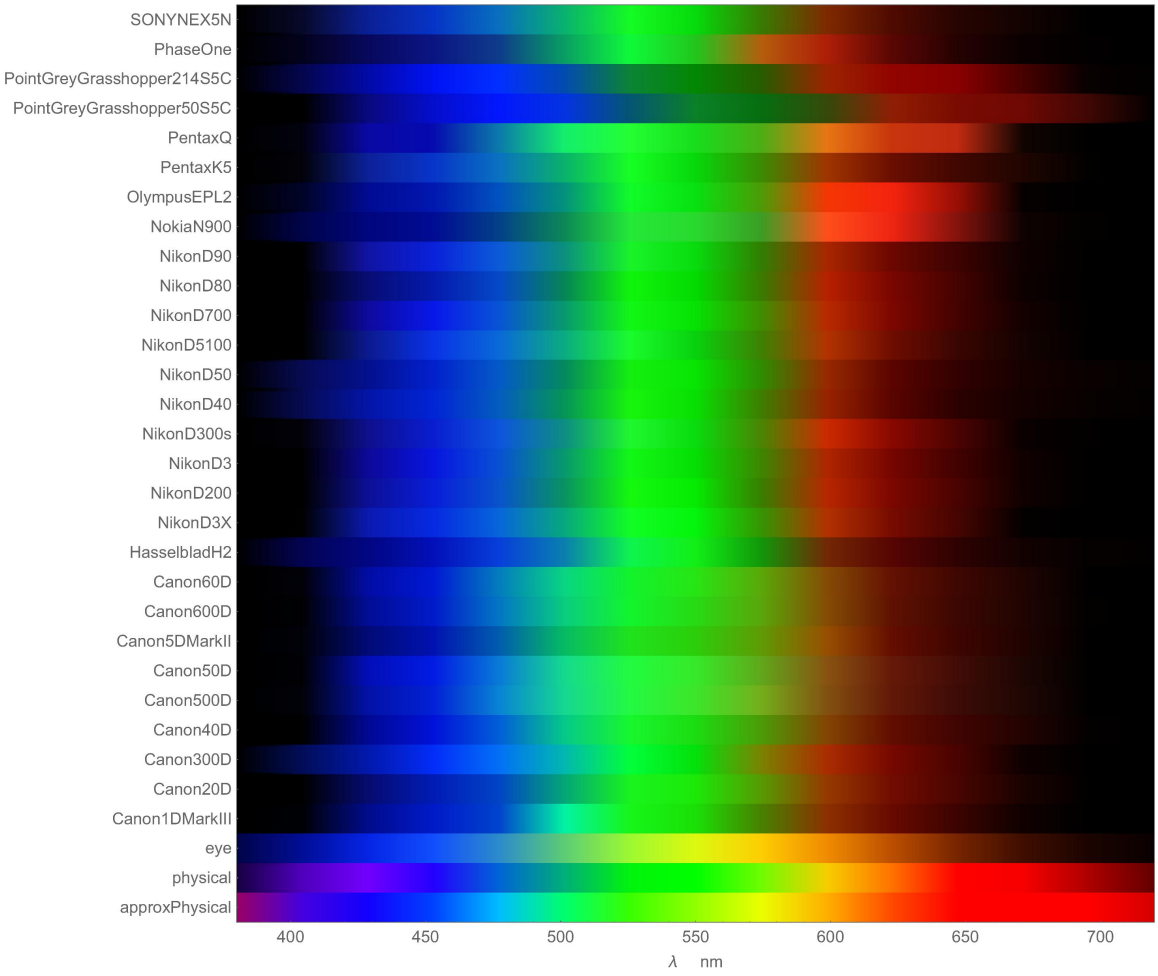


Fig. 1.3 The light spectrum as viewed using various CCDs with a physical spectrum for comparison.

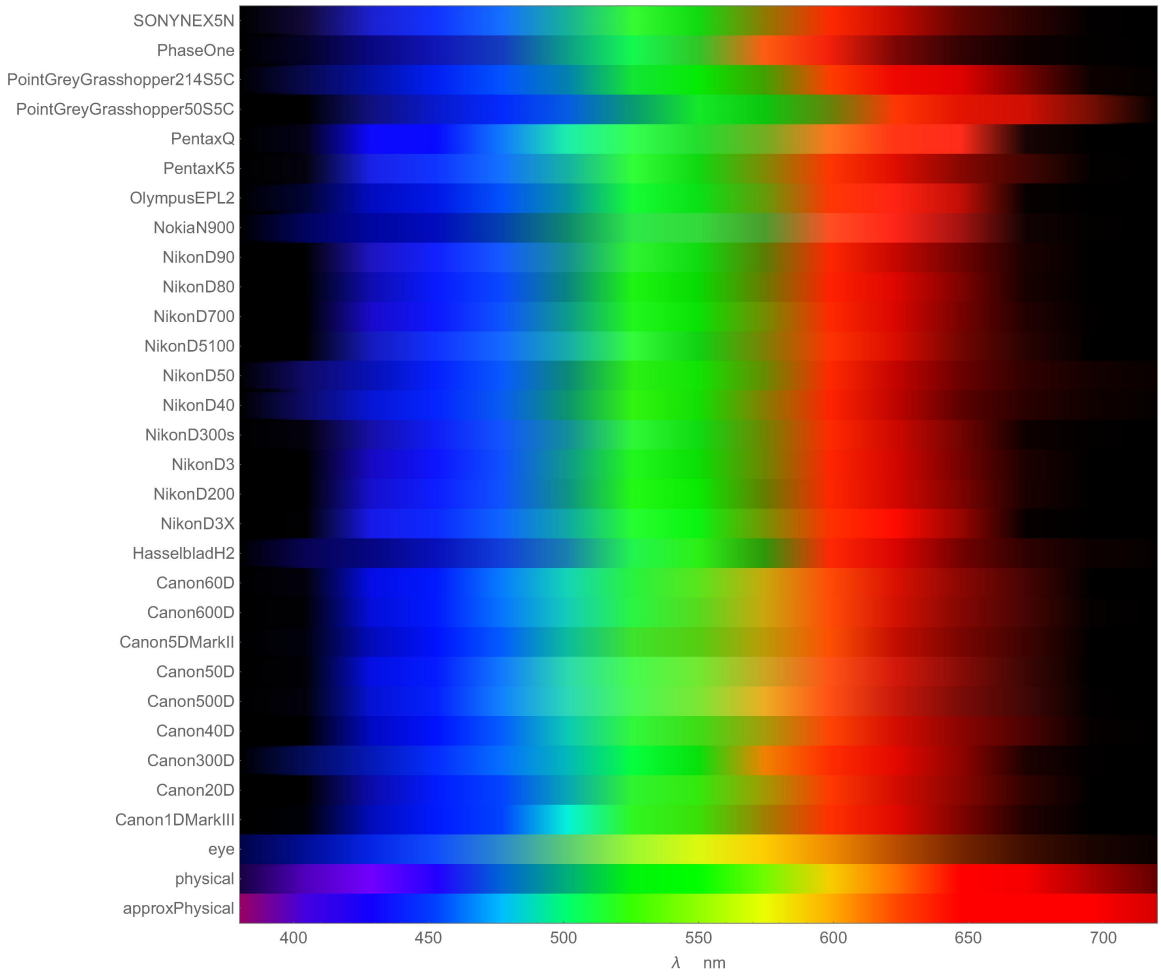


Fig. 1.4 The light spectrum as viewed using various CCDs after normalization with a physical spectrum for comparison.

In addition to the color rebalancing, which corrects the behavior of the CCD devices, as is found in later sections, some devices, like the iPhone, also apply a dynamic color rebalancing which attempts to even out the colors based on the image captured. This feature is intended to correct for strange lighting conditions and other such problems which could affect the image. This will be addressed in detail in Chapter 2.

1.3.3 Skin Color Simulation

So, we have the molar absorptivity for the four key chromophores in the skin, we have the response spectra for a variety of devices, and a response spectra which serves as a target color output for all the devices after the color correction has been applied. Additionally, we are using the CIE 1931 RGB color space for the absorption spectra, which is closer to how our eyes perceive color than any other color space (**?**). This means that, in principle, we can get appropriate RGB values for the four key chromophores at the hardware layer using the device response spectra, or at the AP layer using the CIE 1931 RGB color space spectrum, assuming the device's color space correction does its job. The question now is how to actually simulate skin color.

Given a remittance spectrum $R(\lambda)$ and a response function $S_{rgb}(\lambda)$ for the sensor, the RGB output values can be calculated by finding the convolution

$$R = \int_0^\infty R(\lambda)S_R(\lambda)\mathbf{d}\lambda \tag{1.8}$$

$$G = \int_0^\infty R(\lambda)S_G(\lambda)\mathbf{d}\lambda \tag{1.9}$$

$$B = \int_0^\infty R(\lambda)S_B(\lambda)\mathbf{d}\lambda \tag{1.10}$$

As for finding the RGB values, the bright color is the response of the CCD. By integrating the following functions in the figure below, we get the channel output seen in Figure 1.5.

Thus, we can now find the channel output of any device for which we have a response function (Figure 1.6).

At the time of development, access to the camera on the iPhone was limited to the AP layer, and so we work with the CIE 1931 RGB color space. The relative position of the chromophores in the chromatic space, i.e. a color space with a luminosity axis, is shown in Figure 1.7.

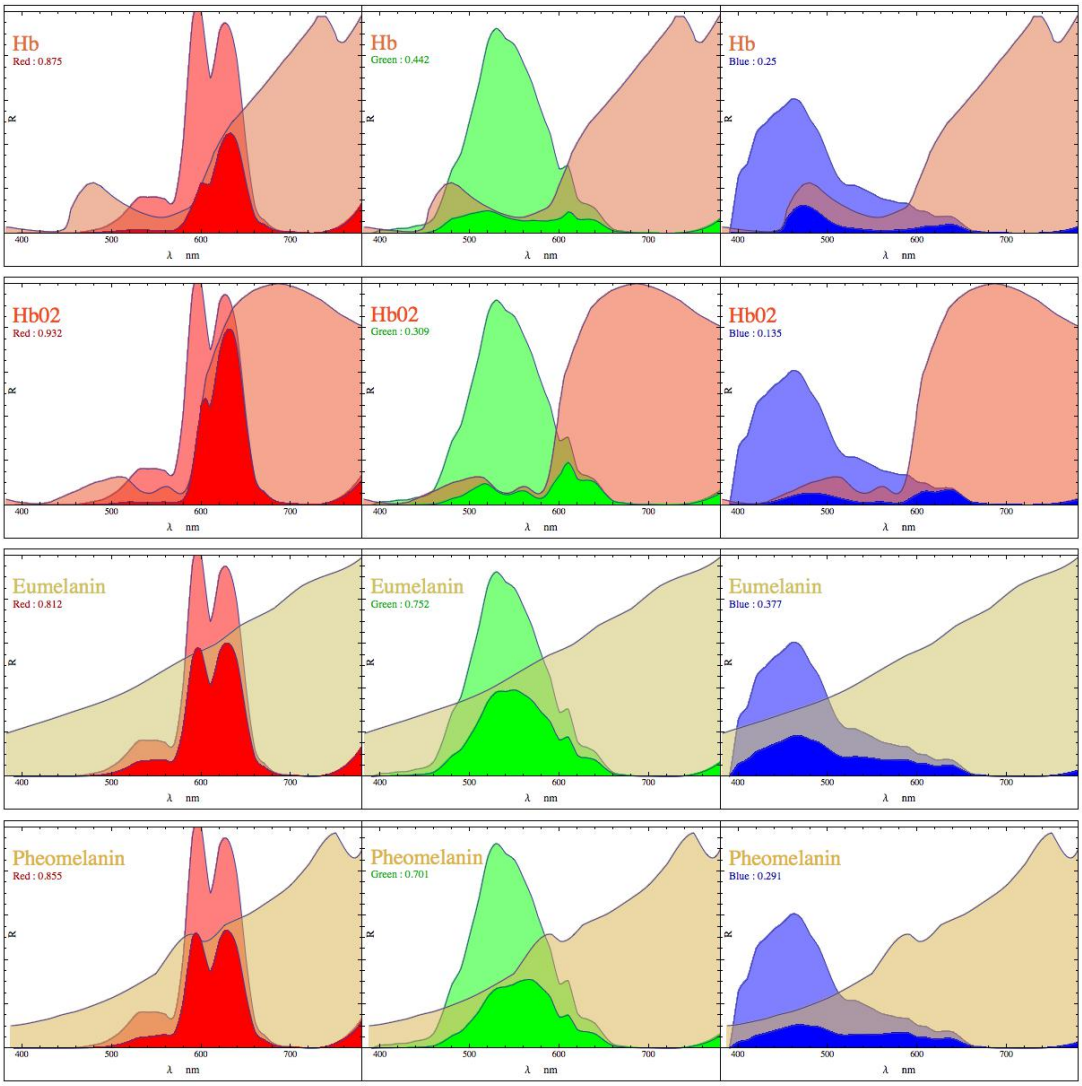


Fig. 1.5 The functions representing the channel output of the four chromophores. The color of the remittance curve is the color for that particular chromophore as seen by the NokiaN900 camera. The solid-filled, bright function is the response of the CCD for each of the channels to that limited light, and the pale function is the response function of the CCD for each of the channels.

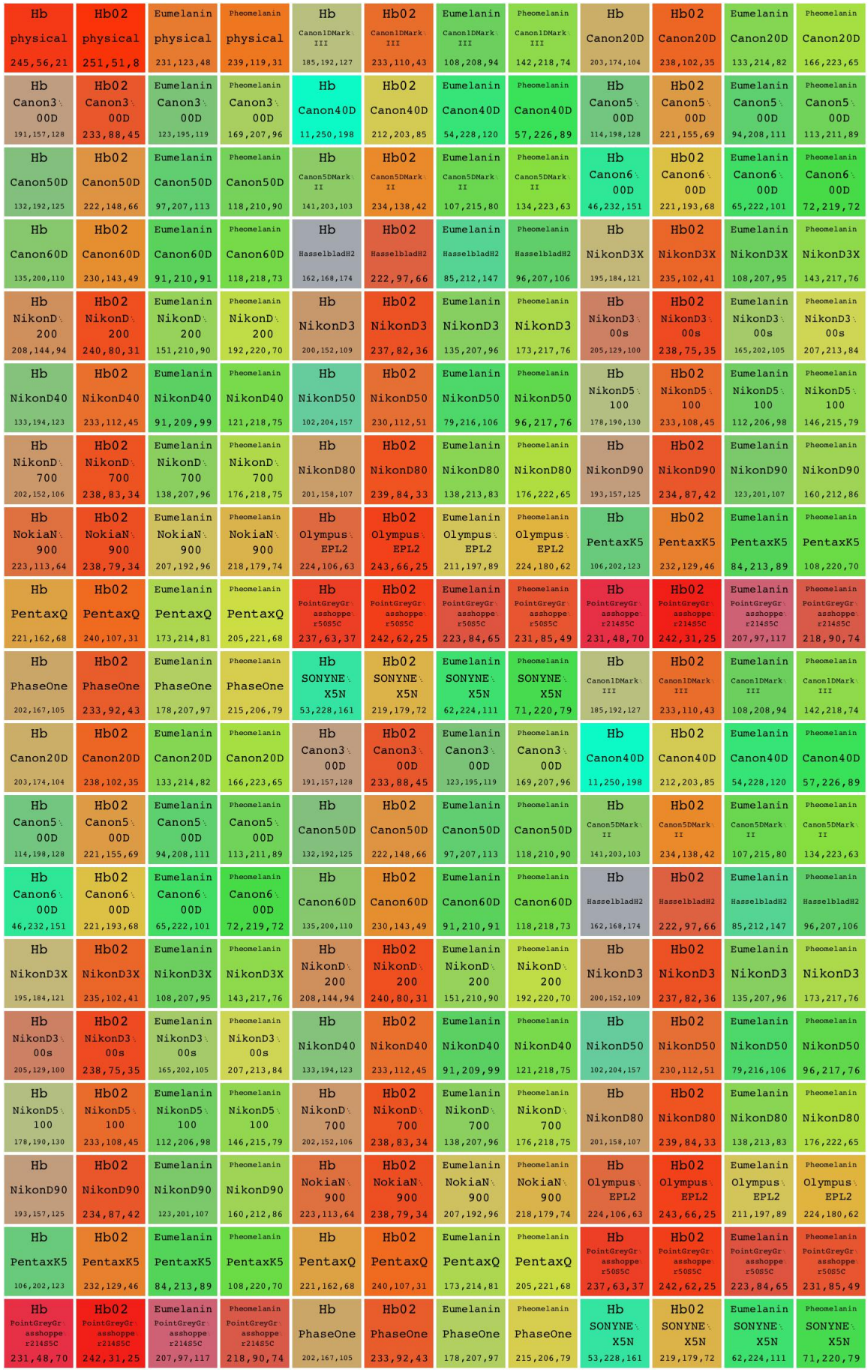


Fig. 1.6 Chromophore colors as perceived by a range of devices.

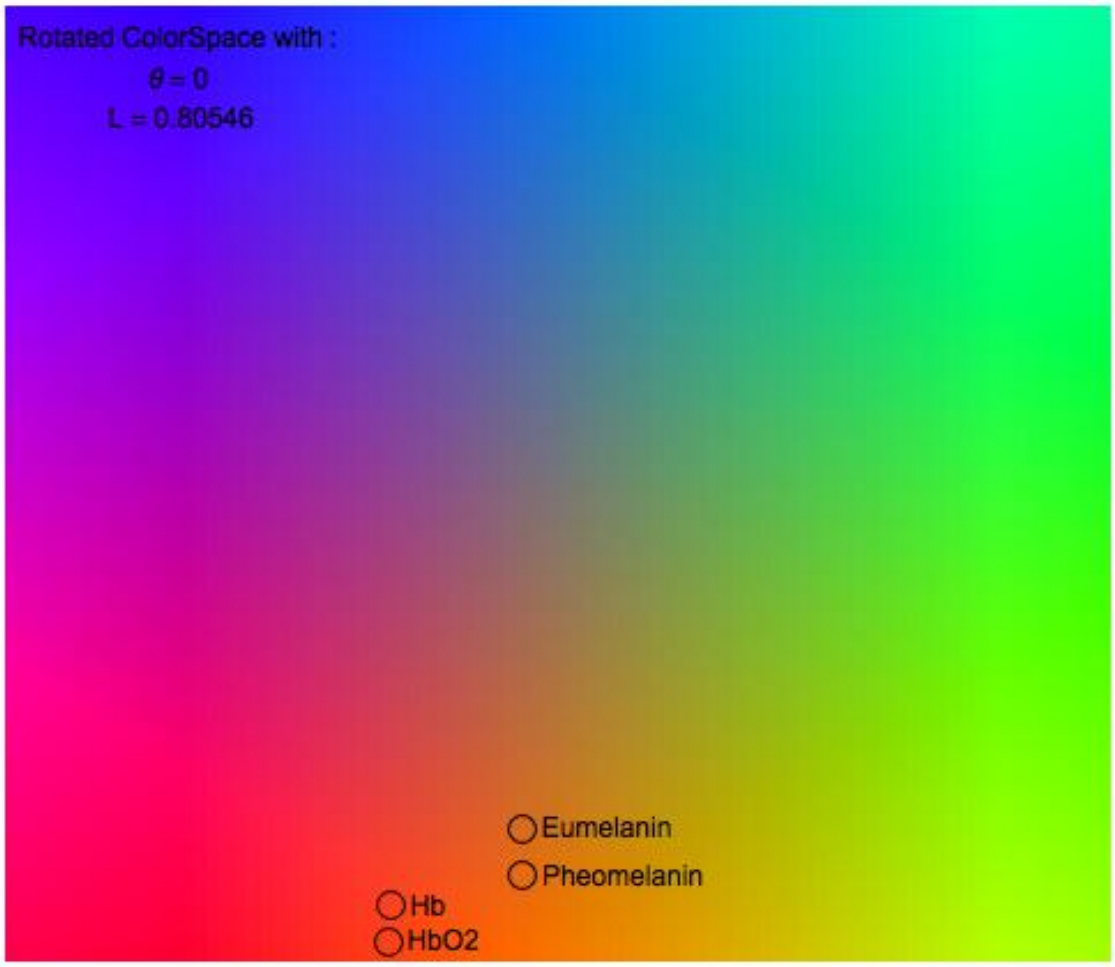


Fig. 1.7 The four main chromophores as perceived by CIE 1931 RGB color space after rotation into the LCaCb $\theta = 0$ color space.

Chapter 2

Color Spaces and Information Storage for Computer Vision Processing

2.1 Constructing a New Color Space

In order to construct a new color space, we need to consider the coordinate system, the orientation, and the fidelity of the discrete representation of the axes.

A Cartesian coordinate system allows for a straightforward transformation from the RGB source involving only rotation, translation and scaling. An orientation with a luminosity axis is useful because we're interested in the color information in the image. This choice determines two of three rotational degrees of freedom, as will be discussed below.

As for the discrete representation of the axes, it's desired that all the information captured pertaining to skin color should be preserved.

2.1.1 Camera RGB and Normalization for Discrete Range

Due to the iPhone hardware being locked down at the application level, we do not have access to the raw camera feed. We do, however, have access to the post-processing (color-rebalanced and white-point-corrected) 8-bit RGB image data. Investigating the difficulties introduced by these processing effects and accounting for them is looked at in a later chapter 3.2.

Computer vision tasks are computationally intensive and often operate in real time. Reduction of the image data set size is a simple method for improving performance. There is, therefore, a need to develop techniques which keep the relevant information while quickly and efficiently discarding the irrelevant information. Achieving this for the RGB space is

the aim of this chapter.

2.2 The Skin Color Space Algorithm

Now that all of the values necessary to preserve the skin information have been obtained, we can use them to build a color space transformation algorithm which can make intelligent decisions about the numerical precision for the intermediate and final variables, as well as determining the most efficient transformation methods. The algorithm described herein will take values of θ, the rotation about the luminosity axis, the standard deviations σ_a,σ_b and mean values μ_a,μ_b for the two chromatic axes in the unit range and will automatically decide upon the necessary intermediate working data types and the most efficient re-distribution methods.

Previously, we found a rotational transformation which allows a working type tRange to be chosen such that tRange $\leq$ 2srcRange. If we were to keep the same data type for the color space dstRange as is used for the RGB values srcRange, then the axes would have to be rescaled with the accompanying loss of information. Given that we have values which allow us to assess where all the relevant information lies, a more sophisticated approach is possible. For a chromatic axis — which, after rotation, has a length $L(\theta)$ — we can determine the positions on that axis at which the information is considered irrelevant using Equation (C.4) and the positions where the information is all considered relevant. If the gradient Δ (C.14) is less than 1, then the distribution loses information at all points on the axis and the axis can be shortened without loss of relevant information. The only further consideration is to ensure that the values outside that range are prevented from causing errors associated with overflow. To exclude this possibility, a conditional statement can be used which checks the bounds as stated, assigning an appropriate value as necessary. The alternative is to use an intermediate value with a higher bit depth, and then to recast into the destination data type in such a way that overflow and underflow are handled appropriately. The OpenCV library provides a casting method — "saturateCast" — which serves this purpose.

2.2.1 Setting the Value for the Tolerance

Now that we have the working type range tRange, accounting for both the rotation and the statistics, we can set a meaningful tolerance on the error. Previously, we calculated that the maximum error would occur for pixel values at the corners of the RGB cube; we have now seen that some values are more important than others and that it is more meaningful

to use a smaller RGB cube which encloses only the values of interest. This cube is found by taking the values for Λ as the corners in the rotated space and rotating back to the RGB space. As this only needs to be done once it can be performed using floats in the unit spaces, however the values of λ account for the discrete numerics in that they are calculated from the discrete values Λ using the new values for κ and tRange. In order to perform the inverse rotation, the values must be shifted to compensate for the natural range of the rotation which is: $\{0:1,-½:½,-½:½\}$.

$$\lambda_2^{RGB} = \overline{\mathbf{R}}^{-1} \cdot (\lambda_2 - \mathbf{R}\mathrm{c}) \qquad \text{where} \quad \mathbf{R}\mathrm{c} = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \tag{2.1}$$

$$= \mathbf{R}^T \cdot \left(\vec{L^{-1}} \otimes (\lambda_2 - \mathbf{R}\mathrm{c}) \right) \tag{2.2}$$

Because rotations of any angle are allowed, the values for λ_1^{RGB} may be larger than those for λ_2^{RGB} in the RGB space. A smart programmer could actually take advantage of this. The values may also be outside the RGB cube, meaning that the values may have to be truncated to fit inside the RGB cube range.

The perturbation to the rotated channel elements $\vec{\delta w}$ is found for an input set of pixel values λ_2^{RGB}

$$\vec{\delta w} = \min\{K\delta(\mu,\sigma)\,,\mathbf{L}(\theta)\} \otimes \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix} \otimes \widehat{\delta\mathbf{q}\mathbf{E}\mathrm{O}} \left[\begin{pmatrix} 0 \\ \delta\mathbf{q}\mathbf{R}\mathrm{e}(\theta_1) \\ \delta\mathbf{q}\mathbf{R}\mathrm{e}\left(\frac{\pi}{6} - \theta_1\right) \end{pmatrix} \right] \otimes \left(\delta\mathbf{q}\mathbf{S}(\theta) \cdot \lambda_2^{RGB} \right)$$

We have previously solved the case where both channels are of equal importance; we now need to find the values of α and β which set the relative importance of each channel. There are two factors here: the channel scaling and the new, smaller RGB cube corners. The values for these also need to be put in correct correspondence with the a and b channel functions. For the scaling this is found by:

$$\begin{pmatrix} 0 \\ \alpha_1 \\ \beta_1 \end{pmatrix} = \widehat{\delta\mathbf{q}\mathbf{E}\mathrm{O}} \left[\min\{K\delta(\mu,\sigma)\,,\mathbf{L}(\theta)\} \otimes \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix} \right]$$

For the new RGB corners we need to find the largest element which can result from the inner product with $\delta\mathbf{qS}(\theta)$. As the elements of $\delta\mathbf{qS}(\theta)$ are in $\{-1,0,1\}$, with only one occurrence of each in the second and third rows, we need to find the largest element of λ_2^{RGB} which is not in the zero position. The algorithm solves:

$$\begin{pmatrix} 0 \\ \alpha_2 \\ \beta_2 \end{pmatrix} = \widehat{\delta\mathbf{qEO}}\left[\max\left\{\text{abs}(\delta\mathbf{qS}(\theta)) \otimes \left(\lambda_2^{RGB}\right)^T\right\}\right]$$

where max acts on each row and $\text{abs}(\delta\mathbf{qS}(\theta)) = \delta\mathbf{qS}(\theta) \otimes \delta\mathbf{qS}(\theta)$. These then give a combined value for $\alpha = \alpha_1\alpha_2$ and $\beta = \beta_1\beta_2$. Because the axis have been scaled so that the information on the axis is to be kept at least near the mean, the tolerance should be set to $\tau = 1$. The condition for accepting a value of θ $1 > h(i,\imath;\alpha,\beta)$ can be used.

2.2.2 The Complete Algorithm

The programmer specifies a requested value of θ, values for the means μ_{ab} and standard deviations σ_{ab} for each of the three axis in the rotated color space with a unit range and the data type ranges srcRange and dstRange. The algorithm then proceeds as follows:

1. Find the working type and range. Using the distribution constants (C.5), the gradient at the mean in the unit range (C.14), and the natural rotated axis lengths (A.7):

$$\delta = \frac{\sqrt{2}}{\sigma\sqrt{\pi}\,(\Sigma^+ - \Sigma^-)} \quad K = \frac{\text{dstRange}}{\text{srcRange}} \quad \Sigma^- = \text{erf}\left(\frac{\mu - 1}{\sqrt{2}\sigma}\right) \quad \Sigma^+ = \text{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)$$

$$\mathbf{L}(\theta) = \begin{pmatrix} \sqrt{3} \\ \sqrt{\frac{2}{3}}\sin\left(\widetilde{\vartheta}\right) + \sqrt{2}\cos\left(\widetilde{\vartheta}\right) \\ \sqrt{\frac{2}{3}}\sin\left(\widetilde{\theta}\right) + \sqrt{2}\cos\left(\widetilde{\theta}\right) \end{pmatrix} \quad \text{where} \quad \begin{matrix} \widetilde{\vartheta} = \left(\theta - \frac{\pi}{6}\right) \bmod \frac{\pi}{3} \\ \widetilde{\theta} = \theta \bmod \frac{\pi}{3} \end{matrix}$$

The length of the axis tRange after rescaling is given by C.16, and the axis limits are easily found along with the compression ratio κ (C.18).

$$\text{tRange}(\theta, \mu, \sigma) = \min\left\{K\delta(\mu, \sigma)\,, \mathbf{L}(\theta)\right\} \text{srcRange} \qquad \kappa(\theta) = \max\left\{\frac{1}{\delta(\mu, \sigma)}, \frac{K}{\mathbf{L}(\theta)}\right\}$$

$$\text{tMin}(\theta, \mu, \sigma) = \begin{pmatrix} 0 \\ \frac{-1}{2}\text{tRange}_2(\theta, \mu_2, \sigma_2) \\ \frac{-1}{2}\text{tRange}_3(\theta, \mu_3, \sigma_3) \end{pmatrix} \qquad \text{tMax}(\theta, \mu, \sigma) = \begin{pmatrix} \text{tRange}_1(\theta, \mu_1, \sigma_1) \\ \frac{1}{2}\text{tRange}_2(\theta, \mu_2, \sigma_2) \\ \frac{1}{2}\text{tRange}_3(\theta, \mu_3, \sigma_3) \end{pmatrix}$$

2. Set the distribution region boundary constants. Using the constants previously defined (C.10 , C.3)

$$w(\mu,\sigma)=\sigma\sqrt{\log\left(\frac{2}{\pi}\right)-2\log\left(\kappa\sigma\left(\Sigma^{+}-\Sigma^{-}\right)\right)}\qquad \mathrm{dL}=\frac{1}{\mathrm{dstRange}}$$

We find the boundaries in the unit range

$$\begin{aligned}\lambda_1&=\sigma\sqrt{2}\,\mathrm{erf}^{-1}\left((\mathrm{dL}-1)\Sigma^{+}-\mathrm{dL}\,\Sigma^{-}\right)+\mu \quad \omega_1(\mu,\sigma)=\mu-w(\mu,\sigma)\\ \lambda_2&=\sigma\sqrt{2}\,\mathrm{erf}^{-1}\left((\mathrm{dL}-1)\Sigma^{-}-\mathrm{dL}\,\Sigma^{+}\right)+\mu \quad \omega_2(\mu,\sigma)=\mu+w(\mu,\sigma)\end{aligned}\tag{2.3}$$

and the boundaries in the working range

$$\Lambda=\mathrm{tMin}+\mathrm{tRange}\left(\lambda(\mu,\sigma)\right)\quad \Omega(\mu,\sigma)=\mathrm{tMin}+\mathrm{tRange}\left(\omega(\mu,\sigma)\right)\tag{2.4}$$

The extended region in which all the information is kept is found by numerically solving the defining equations for the extended boundaries Ωp_1 and Ωp_2 with the practical introduction of a parameter τ_p, which extends the acceptable deviation from linearity. So $\tau_p=1$ would allow 1 discarded piece of information within the extended region.

$$\lceil \mathrm{dis}\left(\Omega_1\right)\rceil-\Omega_1-\tau_p=\mathrm{dis}\left(\Omega p_1\right)-\Omega p_1\quad \lfloor \mathrm{dis}\left(\Omega_2\right)\rfloor-\Omega_2-\tau_p=\mathrm{dis}\left(\Omega p_2\right)-\Omega p_2\tag{2.5}$$

The C++ code finds the points using the following algorithm

Algorithm 1 Finding the extended region in which all the information is kept in C++

Require: Ω_2 the starting point for the linear walk
τ_p the acceptable divergence from linearity
Ensure: $\{\Omega p_1,\Omega p_2\}$ the extended boundaries
$x\leftarrow\Omega_2$ ▷ $dis(x)$ is the distribution function
while $dis(x)>x+\lfloor \mathrm{dis}\left(\Omega_2\right)\rfloor-\Omega_2-\tau_p$ **do**
x++
end while
$\Omega p_2\leftarrow x$
$\Omega p_1\leftarrow\Omega_1-(\Omega p_2-\Omega_2)$ ▷ Exploiting a symetry dis(x)
Return Ωp_1 ,Ωp_2

The unit range boundaries ωp are found using $\omega p=\frac{\Omega p-\mathrm{tMin}}{\mathrm{tRange}}$.

3. Suggest a new value for θ. First we find the perturbation scaling (B.6) and the perturbation function ordering function (**??**). In order to set the values for the scaling of the perturbation α and β, the absolute value of the perturbation matrix sign $\|\delta\mathbf{qS}(\theta)\|$ can be written in terms of a single row vector function $\delta\mathbf{qAS}(\theta)$

$$\|\delta\mathbf{qS}(\theta)\| = \begin{pmatrix} 0 & 0 & 0 \\ \cdots & \delta\mathbf{qAS}\left(\theta+\frac{\pi}{2}\right) & \cdots \\ \cdots & \delta\mathbf{qAS}(\theta) & \cdots \end{pmatrix} \qquad \delta\mathbf{qAS}(\theta) = \begin{cases} \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} & 0 \leq \ (\theta \bmod \pi) \ < \frac{\pi}{3} \\ \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} & \frac{\pi}{3} \leq \ (\theta \bmod \pi) \ < \frac{2\pi}{3} \\ \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} & \frac{2\pi}{3} \leq \ (\theta \bmod \pi) \ < \pi \end{cases}$$

Then we find the point at which loss of information does not matter in RGB space λ_2^{RGB}

$$\lambda_2^{RGB} = \max\left\{\mathbf{R}^T \cdot \left(\vec{L^{-1}} \otimes (\lambda_1 - \mathbf{R}\mathrm{c})\right), \mathbf{R}^T \cdot \left(\vec{L^{-1}} \otimes (\lambda_2 - \mathbf{R}\mathrm{c})\right)\right\} \quad \text{where} \quad \mathbf{R}\mathrm{c} = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

Then we numerically find

$$\begin{pmatrix} 0 \\ \alpha \\ \beta \end{pmatrix} = \widehat{\delta\mathbf{qEO}}\left[\begin{pmatrix} 0 \\ 2\min\left\{K\delta(\mu_2,\sigma_2), \mathbf{L}_2(\theta)\right\}\max\left\{\delta\mathbf{qAS}\left(\theta+\frac{\pi}{2}\right)\otimes\left(\lambda_2^{RGB}\right)^T\right\} \\ 2\min\left\{K\delta(\mu_3,\sigma_3), \mathbf{L}_3(\theta)\right\}\max\left\{\delta\mathbf{qAS}(\theta)\otimes\left(\lambda_2^{RGB}\right)^T\right\} \end{pmatrix}\right]$$

The algorithm then finds a starting value for the index (B.15) and returns the closest value which satisfies the tolerance (B.21):

$$i(\delta\theta, n) = \frac{\bar{\mathbf{i}}\tan(\delta\theta)\left(\sqrt{3}\tan(\delta\theta)+7\right)}{\tan(\delta\theta)+\sqrt{3}} \qquad \delta\theta = \theta \mod \frac{\pi}{6} \qquad \bar{\mathbf{i}} = 2^{n-3}$$

The region ι for a given value of i is found and the appropriate values (B.17) for the extrema (B.12) are evaluated in the function which finds the perturbation at the index (B.21) and the intercept position $\overset{\times}{\phi}(i)$ is found with algorithm 5.

To find the angular value closest to θ which produces a perturbation less than the tolerance τ, algorithm 6 finds the closest value which satisfies the tolerance in the positive and negative directions and returns the nearest one to the requested value of θ. Algorithm 6 is guaranteed to find a value which satisfies the tolerance in each direction within a $\frac{\pi}{6}$ region because the perturbation is zero at the ends of the $\frac{\pi}{6}$ region.

Algorithm 2 A function which returns the angular position of compromise between the perturbations to the channels.

function $\overset{\times}{\phi}(i,\alpha,\beta,n)$ ▷ i integer index of the intercept

$\bar{\mathbf{i}} \leftarrow 2^{n-3}$; $\quad \imath(i,n) \leftarrow \left\lfloor 7\,\bar{\mathbf{i}} - \sqrt{i^2 - 2i\bar{\mathbf{i}} + 49\,\bar{\mathbf{i}}^2} \right\rfloor$ ▷ n the source bit depth

$\mathring{\phi}_b(\mathbf{l}) = \arctan\left(\frac{\mathbf{l}2^{2-n}}{\sqrt{3}}\right)$; $\quad \mathring{\phi}_a(\mathbf{l}) = \arctan\left(\frac{\mathbf{l}\sqrt{3}}{2^n - \mathbf{l}}\right)$

if $i+\imath \bmod 2 = 0$ **then** ▷ $i+\imath$ is even

$\hat{\phi}_a \leftarrow \mathring{\phi}_a(i+\imath+1)$; $\quad \check{\phi}_a \leftarrow \mathring{\phi}_a(i+\imath)$

$\hat{\phi}_b \leftarrow \mathring{\phi}_b(i-\imath-1)$; $\quad \check{\phi}_b \leftarrow \mathring{\phi}_b(i-\imath)$

else

$\hat{\phi}_a \leftarrow \mathring{\phi}_a(i+\imath)$; $\quad \check{\phi}_a \leftarrow \mathring{\phi}_a(i+\imath+1)$

$\hat{\phi}_b \leftarrow \mathring{\phi}_b(i-\imath)$; $\quad \check{\phi}_b \leftarrow \mathring{\phi}_b(i-\imath-1)$

end if

$\overset{\times}{\phi}(i) \leftarrow \left\{ \frac{\beta\check{\phi}_b(\check{\phi}_a-\hat{\phi}_a)+\alpha\check{\phi}_a(\hat{\phi}_b-\check{\phi}_b)}{\beta(\check{\phi}_a-\hat{\phi}_a)+\alpha(\hat{\phi}_b-\check{\phi}_b)}, \frac{\alpha\beta(\check{\phi}_a-\check{\phi}_b)}{\beta(\check{\phi}_a-\hat{\phi}_a)+\alpha(\hat{\phi}_b-\check{\phi}_b)} \right\}$

Return $\overset{\times}{\phi}(i)$ ▷ $\overset{\times}{\phi}(i)$ {angle, maximum perturbation}

end function

Algorithm 3 Suggest a new value for θ

Require: α, β the relative importance of the channels

θ the requested angle; $\quad \tau$ the tolerance

Ensure: ϑ the suggested value for θ and h the predicted maximum perturbation.

$\delta\theta \leftarrow \theta \bmod \frac{\pi}{6}$; $\quad \Theta \leftarrow \theta - \delta\theta$; $\quad \bar{\mathbf{i}} \leftarrow 2^{n-3}$;

$i_\ominus \leftarrow i_\oplus \leftarrow \frac{\bar{\mathbf{i}}\tan(\delta\theta)\left(\sqrt{3}\tan(\delta\theta)+7\right)}{\tan(\delta\theta)+\sqrt{3}}$; $\quad \{\vartheta_\ominus, h_\ominus\} \leftarrow \{\vartheta_\oplus, h_\oplus\} \leftarrow \overset{\times}{\phi}(i,\alpha,\beta,n)$

while $h_\oplus > \tau$ **do**

$i_\oplus{+}{+}$

$\{\vartheta_\oplus, h_\oplus\} \leftarrow \overset{\times}{\phi}(i_\oplus,\alpha,\beta,n)$

end while

while $h_\ominus > \tau$ **do**

$i_\ominus{-}{-}$

$\{\vartheta_\ominus, h_\ominus\} \leftarrow \overset{\times}{\phi}(i_\ominus,\alpha,\beta,n)$

end while

if $\vartheta_\oplus - \delta\theta < \delta\theta - \vartheta_\ominus$ **then**

$\{\vartheta, h\} = \{\Theta + \vartheta_\oplus, h_\oplus\}$

else

$\{\vartheta, h\} = \{\Theta + \vartheta_\ominus, h_\ominus\}$

end if

Return $\{\vartheta, h\}$

4. Set the integer rotation matrix.

Expanding the definition of the quantized matrix $\mathbf{qR}(\theta,n)$ (B.2) with the definitions of the factored matrix $\mathbf{fR}(\theta)$ (A.16), the matrix ordering $\mathbf{fRO}[\cdots,\theta]$ (Table A.2) and the factored matrix function $\mathbf{fRe}(\phi)$ (A.15) allows a more algorithm minded form to be written.

$$\mathbf{qRs} = f\vec{S}s \otimes \mathbf{qR} \quad \mathbf{U} = 2^{n-2} \quad \theta_6 = \theta \bmod \pi \quad \theta_1 = \theta \bmod \frac{\pi}{6}$$

$$\mathbf{A}_{\oplus} = -\left(2^{n-3} + \text{Round}\left[2^{n-3}\sqrt{3}\tan(\theta_1)\right]\right) \qquad -2^{n-2} \leq \mathbf{A}_{\oplus} \leq -2^{n-3}$$

$$\mathbf{A}_{\ominus} = -\left(2^{n-3} - \text{Round}\left[2^{n-3}\sqrt{3}\tan(\theta_1)\right]\right) \qquad -2^{n-3} \leq \mathbf{A}_{\ominus} \leq 0$$

$$\mathbf{B}_{\oplus} = -\left(2^{n-3} + \text{Round}\left[2^{n-3}\sqrt{3}\tan\left(\frac{\pi}{6} - \theta_1\right)\right]\right) \qquad -2^{n-2} \leq \mathbf{B}_{\oplus} \leq -2^{n-3}$$

$$\mathbf{B}_{\ominus} = -\left(2^{n-3} - \text{Round}\left[2^{n-3}\sqrt{3}\tan\left(\frac{\pi}{6} - \theta_1\right)\right]\right) \qquad -2^{n-3} \leq \mathbf{B}_{\ominus} \leq 0$$

$$\mathbf{qR}(\theta,n) = \begin{cases} \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{A}_{\oplus} & \mathbf{U} & \mathbf{A}_{\ominus} \\ \mathbf{B}_{\oplus} & \mathbf{B}_{\ominus} & \mathbf{U} \end{pmatrix} & 0 \leq \theta_6 < \frac{\pi}{6} \\ \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{U} & \mathbf{B}_{\oplus} & \mathbf{B}_{\ominus} \\ \mathbf{A}_{\ominus} & \mathbf{A}_{\oplus} & \mathbf{U} \end{pmatrix} & \frac{\pi}{6} \leq \theta_6 < \frac{\pi}{3} \\ \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{U} & \mathbf{A}_{\ominus} & \mathbf{A}_{\oplus} \\ \mathbf{B}_{\ominus} & \mathbf{U} & \mathbf{B}_{\oplus} \end{pmatrix} & \frac{\pi}{3} \leq \theta_6 < \frac{\pi}{2} \\ \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{B}_{\oplus} & \mathbf{B}_{\ominus} & \mathbf{U} \\ \mathbf{A}_{\oplus} & \mathbf{U} & \mathbf{A}_{\ominus} \end{pmatrix} & \frac{\pi}{2} \leq \theta_6 < \frac{2\pi}{3} \\ \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{A}_{\ominus} & \mathbf{A}_{\oplus} & \mathbf{U} \\ \mathbf{U} & \mathbf{B}_{\oplus} & \mathbf{B}_{\ominus} \end{pmatrix} & \frac{2\pi}{3} \leq \theta_6 < \frac{5\pi}{6} \\ \begin{pmatrix} 1 & 1 & 1 \\ \mathbf{B}_{\ominus} & \mathbf{U} & \mathbf{B}_{\oplus} \\ \mathbf{U} & \mathbf{A}_{\ominus} & \mathbf{A}_{\oplus} \end{pmatrix} & \frac{5\pi}{6} \leq \theta_6 < \pi \end{cases} \qquad f\vec{S}s(\theta) = \begin{cases} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & 0 \leq \theta_6 < \frac{\pi}{6} \\ \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} & \frac{\pi}{6} \leq \theta_6 < \frac{\pi}{3} \\ \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} & \frac{\pi}{3} \leq \theta_6 < \frac{\pi}{2} \\ \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} & \frac{\pi}{2} \leq \theta_6 < \frac{2\pi}{3} \\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & \frac{2\pi}{3} \leq \theta_6 < \frac{5\pi}{6} \\ \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} & \frac{5\pi}{6} \leq \theta_6 < \pi \end{cases}$$

This $\mathbf{qRs}(\theta,n)$ matrix is used to perform the rotation of the pixel values. Taking the inner product of $\mathbf{qRs}$ with the pixel values produces a rotated set of pixel values $\vec{a} = \mathbf{qRs} \cdot \vec{rgb}$ with elements $\vec{a}_1 \in \{0 \cdots 3(2^n-1)\}$, $\vec{a}_2 \in 2^{n-2} - 2^{2n-2} \cdots 2^{2n-2} - 2^{n-2}\}$ and $\vec{a}_3 \in 2^{n-2} - 2^{2n-2} \cdots 2^{2n-2} - 2^{n-2}\}$.

$$\text{qRsMin} = \begin{pmatrix} 0 \\ 2^{n-2} - 2^{2n-2} \\ 2^{n-2} - 2^{2n-2} \end{pmatrix} \qquad \text{qRsMax} = \begin{pmatrix} 3(2^n-1) \\ 2^{2n-2} - 2^{n-2} \\ 2^{2n-2} - 2^{n-2} \end{pmatrix} \qquad \text{qRsRange} = \begin{pmatrix} 3(2^n-1) \\ 2^{n-1}(2^n-1) \\ 2^{n-1}(2^n-1) \end{pmatrix}$$

5. Design the distribution function. First, the overall scaling for the quantized rotation matrix is found by combining the normalized scaling $\vec{S}(\theta, n)$ (**??**) with the desired scaling defined in equation (C.16):

$$\vec{S} = \min\left\{K\delta(\mu,\sigma)\,, \mathbf{L}(\theta)\right\} \otimes \begin{pmatrix} \frac{1}{3} \\ 2^{1-n} \\ 2^{1-n} \end{pmatrix}$$

Next, we decide which of the distribution methods should be used. This is done by assessing the size of each of the regions relative to a tolerance. The algorithm avoids unnecessary operations by finding the region directly with the unscaled result of the inner product between the matrix **qRs** and the pixel value.

Algorithm 4 An algorithm which returns a channel appropriate distribution function

Require: $\{\lambda, \Lambda, \omega p, \Omega p\}$ the boundaries of the regions.
$0 < \tau < 1$ the tolerances for counting a region as significant.

Ensure: *pDis* a function which performs the re-distribution for a channel.

$\mathbf{Q}_{discard} \leftarrow \tau_{discard} \leq 1 - \lambda_2 + \lambda_1$ $\quad$ $\mathbf{Q}_{keep} \leftarrow \tau_{keep} \leq \omega p_2 - \omega p_1$

$\mathbf{Q}_{distribute} \leftarrow \tau_{distribute} \leq \lambda_2 - \omega p_2 + \omega p_1 - \lambda_1$

$\Lambda_1^q \leftarrow \text{qRsRange}\ \lambda_1 + \text{qRsMin};$ $\quad$ $\Lambda_2^q \leftarrow \text{qRsRange}\ \lambda_2 + \text{qRsMin}$

$\Omega p_1^q \leftarrow \text{qRsRange}\ \omega p_1 + \text{qRsMin};$ $\quad$ $\Omega p_2^q \leftarrow \text{qRsRange}\ \omega p_2 + \text{qRsMin}$

if $\mathbf{Q}_{keep} \wedge \mathbf{Q}_{discard} \wedge \mathbf{Q}_{distribute}$ **then** ▷ Piecewise Erf Distribution

$$pDis(x) \leftarrow \begin{cases} \text{dstMin} & x \leq \Lambda_1^q \\ \text{dis}(\vec{S}x) & \Lambda_1^q < x < \Omega p_1^q \\ \vec{S}x - \Omega p_1 + \text{dis}(\Omega p_1) & \Omega p_1^q \leq x \leq \Omega p_2^q \\ \text{dis}(\vec{S}x) + \Omega p_2 - \Omega p_1 - \text{dis}(\Omega p_2) + \text{dis}(\Omega p_1) & \Omega p_2^q < x < \Lambda_2^q \\ \text{dis}(\Lambda_2) + \Omega p_2 - \Omega p_1 - \text{dis}(\Omega p_2) + \text{dis}(\Omega p_1) & \Lambda_2^q \leq x \end{cases}$$

else if $\mathbf{Q}_{keep} \wedge \mathbf{Q}_{discard} \wedge \neg\mathbf{Q}_{distribute}$ **then** ▷ Partitioning Distribution

$$pDis(x) \leftarrow \begin{cases} \text{dstMin} & x \leq \Lambda_1^q \\ \vec{S}\,x + \text{dstMin} & \Lambda_1^q \leq x \leq \Lambda_2^q \\ \Lambda_2 - \Lambda_1 + \text{dstMin} & \Lambda_2^q \leq x \end{cases}$$

Algorithm 4 Part 2

else if $\mathbf{Q}_{keep} \wedge \neg\mathbf{Q}_{discard} \wedge \mathbf{Q}_{distribute}$ **then** ▷ Little Loss Distribution

$$pDis(x) \leftarrow \begin{cases} \mathrm{dis}(\vec{S}x) & \mathrm{srcMin} < x < \Omega p_1^q \\ \vec{S}x - \Omega p_1 + \mathrm{dis}(\Omega p_1) & \Omega p_1^q \leq x \leq \Omega p_2^q \\ \mathrm{dis}(\vec{S}x) + \Omega p_2 - \Omega p_1 - \mathrm{dis}(\Omega p_2) + \mathrm{dis}(\Omega p_1) & \Omega p_2^q < x \end{cases}$$

else if $\mathbf{Q}_{keep} \wedge \neg\mathbf{Q}_{discard} \wedge \neg\mathbf{Q}_{distribute}$ **then** ▷ Linear Distribution

$pDis(x) \leftarrow \vec{S}\,x + \mathrm{dstMin}$

else if $\neg\mathbf{Q}_{keep} \wedge \mathbf{Q}_{discard} \wedge \mathbf{Q}_{distribute}$ **then** ▷ Truncated Erf Distribution

$$pDis(x) \leftarrow \begin{cases} \mathrm{dstMin} & x \leq \Lambda_1^q \\ \mathrm{dis}(\vec{S}x) & \Lambda_1^q < x < \Lambda_2^q \\ \mathrm{dstMax} & \Lambda_2^q \leq x \end{cases}$$

else if $\neg\mathbf{Q}_{keep} \wedge \mathbf{Q}_{discard} \wedge \neg\mathbf{Q}_{distribute}$ **then** ▷ Step Distribution!

$$pDis(x) \leftarrow \begin{cases} \mathrm{dstMin} & x \leq \frac{\Lambda_1^q + \Lambda_2^q}{2} \\ 1 + \mathrm{dstMin} & \frac{\Lambda_1^q + \Lambda_2^q}{2} < x \end{cases}$$

else ▷ $\neg\mathbf{Q}_{keep} \wedge \neg\mathbf{Q}_{discard} \wedge \neg\mathbf{Q}_{distribute}$

Error; ▷ The tolerances or something else must be wrong!

end if

Return $pDis(x)$

2.2.3 Conclusion

The matrix **qRs** achieves its stated aims of being of integer type and — when acting upon vectors of the source type — producing results which fit into a data type of twice the bit depth, but no more. For common source types, the suggested quantized value of θ will likely be within 1 degree of the requested value, and so there are no unacceptable restrictions on the color space. And the re-distribution from the working type to the destination type is performed efficiently with the optimization decisions taken by the code itself. The algorithm thus described therefore only requires the color statistics for the object of interest and the orientation of the color space to be provided by the user, along with optional tolerances variously described above.

During the development of this algorithm, it became apparent with certain coincidences that the same formulations of the rotation would be happened upon in pursuit of different goals. Most notably, the matrix **fR**e — as well as satisfying the optimization goals — also is the smallest representation which provides a truly lossless rotation. This may be surprising because the matrix **R** theoretically provides sufficient space for all the information to be retained. However, there are unavoidable discretization errors which make some values inside the rotated space inaccessible and others 2 : 1 from the RGB space to the LCaCb space. So,

to be clear, **fR**e is the smallest representation which has no 2 : 1 loss of information, but has inaccessible values, whereas LCaCb is the most compact representation which has, at worst, 2 : 1 errors.

It should be noted that neither the inaccessible value problem or the 2 : 1 loss of information is unique to the factored or discretized representation and is as much of a problem even in a floating-point representation of the rotation. The common implementation is to take a source integer, turn it into a floating-point number, perform the conversion in floating points, then turn it back into an integer; this is the case in OpenCV and other such computer vision libraries. These implementations also suffer from the same problem. Hereafter, the problem of inaccessible values and 2 : 1 allocation values due to the discretization will be referred to as "the speckling problem".

Chapter 3

Skin Statistics

3.1 Objective

We are trying to describe monochromatic objects which have a small variation around an average color — the implicit assumption here is that skin is, essentially, monochromatic. Chromatic skin statistics have been collected by several authors (**???????**) using a variety of different approaches, but most have found that the statistics are well-described by 2D Gaussians in the chromatic plane. The goal of this chapter is to find a value of σ (two standard deviations); μ, the position for the mean; and θ', the angle of the major axis of the 2D Gaussian found in the LCaCb $\theta = 0$ color space. This will allow us to use the methods presented in Chapter 2 to design a bespoke color space which will preserve all the chromatic information about the target model.

3.2 iPhone Camera Characteristics

There are two aspects of the camera which affect the collection of the color statistics, the CCD sensitivity as detailed in Chapter 1 and the processing of the detected raw channel values before the AP layer. Unfortunately neither the details of the CCD nor the action of the preprocessing are publicly documented. The relevant effects are discovered and described in this section

3.2.1 The White Point

The iPhone's camera pre-processes the raw image before it reaches the AP layer which causes difficulties in collecting reliable statistics. Typically, cameras have a specific white

point value, which is the RGB value corresponding to white which isn't necessarily the corner of the RGB cube; finding this value is part of the camera calibration. It also determines the orientation of the luminosity axis, which passes through zero to the white point. This is why pre-defined color space functions have an implicit white point correction.

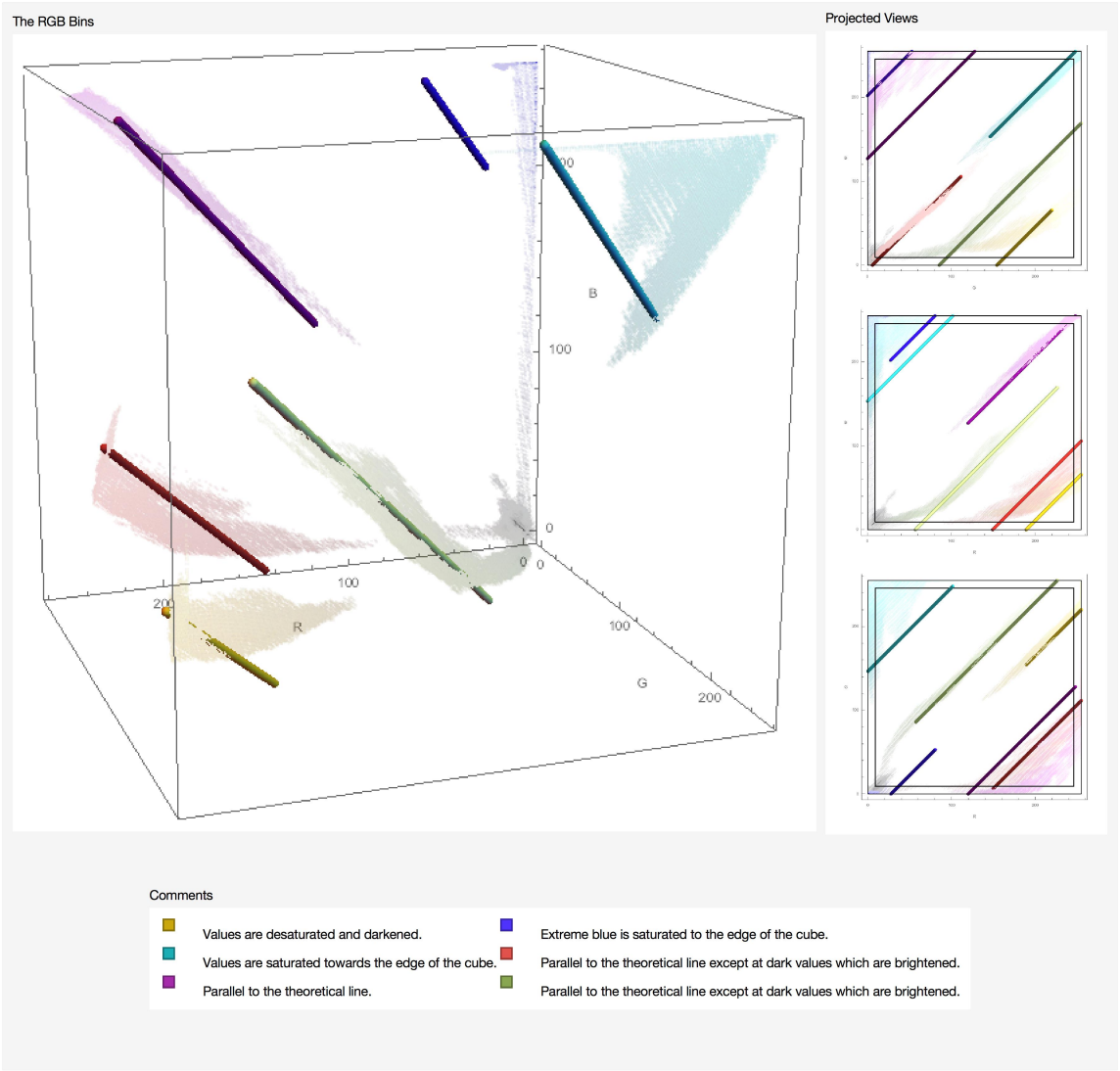


Fig. 3.1 The Bins for 6 randomly colored blobs varying only in luminosity with lines indicating the source image colors.

The white point for the iPhone camera was found by taking a random set of 6 colors $X = \{\{X_R^1 + \lambda, X_G^1 + \lambda, X_B^1 + \lambda\}, \cdots\}$. and varying the luminosity between $\lambda(X)_B < \lambda < \lambda(X)_W$. This produces a set of 6 colors which vary only in luminosity and which do not suffer from white-out or black-out. Panels of these 6 colors with randomly selected luminosity for each were presented to the camera. This produced a set of images and the statistics were

collected for this set. Theoretically we expect there to be 6 distinct parallel lines which are also parallel to the line from black to the white point of the camera. If the white point of the camera is not $\{255, 255, 255\}$ then this test should allow the determination of the real white point.

The iPhone's white point is always set to the corner of the RGB cube before the image reaches the AP layer. So, when developing an algorithm for the iPhone, white point correction is not necessary, while on other devices the algorithm may need to be adapted accordingly.

3.2.2 Color Correction Preprocessing

A second undocumented pre-processing stage became apparent while gathering skin statistics. The original approach taken used a set of photos of an individual's skin under different lighting conditions against a constant monochromatic background captured with the iPhone camera. The background was included in order to obtain data on the edges of the skin. A background set of photos captured under the same conditions but without the presence of a hand would then be used to produce a statistical model which would be used to negate bin counts from the individual's skin set which corresponded to background values.

Surprisingly, this approached failed to work; the background was not represented well by the collected background statistics (Figure 3.2).

The background statistics changed with the skin present in the photo. This is because the iPhone adjusts to images with very strong color characteristics. This color correction is an undocumented feature of the iPhone processing. The only way to compensate for this unwelcome pre-processing is to photograph the background with a strongly contrasting object present, but one which is easily cropped out of the image before the background statistics are collected. After collecting the statistics again, the result was much improved (Figure 3.3).

In a real world context, it is relatively safe to presume that the scene will be chromatically complex enough that the color correction won't be detrimental to detection, and perhaps even beneficial under unusual lighting conditions. But for gathering statistics, it proved to be a massive pain.

3.2.3 White-Out and Black-Out

A monochromatic object with average RGB color $\mu = \{\mu_R, \mu_G, \mu_B\}$ under different lighting conditions produce an in-camera value of $\{\mu_R + \lambda, \mu_G + \lambda, \mu_B + \lambda\}$. Each channel has a

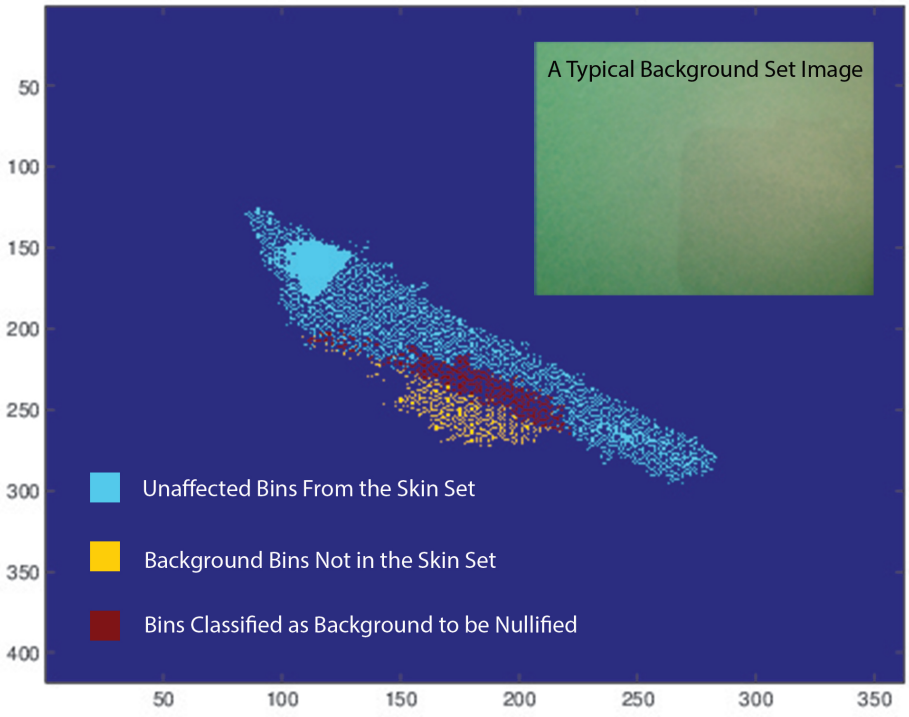


Fig. 3.2 Initial attempt at removing background; unsuccessful.

numerical limit which means that when that limit is reached further change in luminosity λ will result in a change in chromatic value. At high luminosities the colors converge on white becoming washed out. This is referred to as white-out. Similarly at low luminosities the colors converge on black which is referred to as black-out.

It is reasonable to assume that pixel values which contain fully saturated channel elements are unreliable as they may be the result of white-out or black-out. Unfortunately the iPhone also performs a auto brightness and contrast adjustment moving pixel values away from the sides of the RGB cube. The result of this processing is that the values with a luminosity above λ_w and below λ_b are unreliable.

3.3 Algorithm for Generating the Model

Here we present the algorithm which is used to generate the chromatic model. It's assumed we have RGB image sets for the target with simple, monochromatic backgrounds.

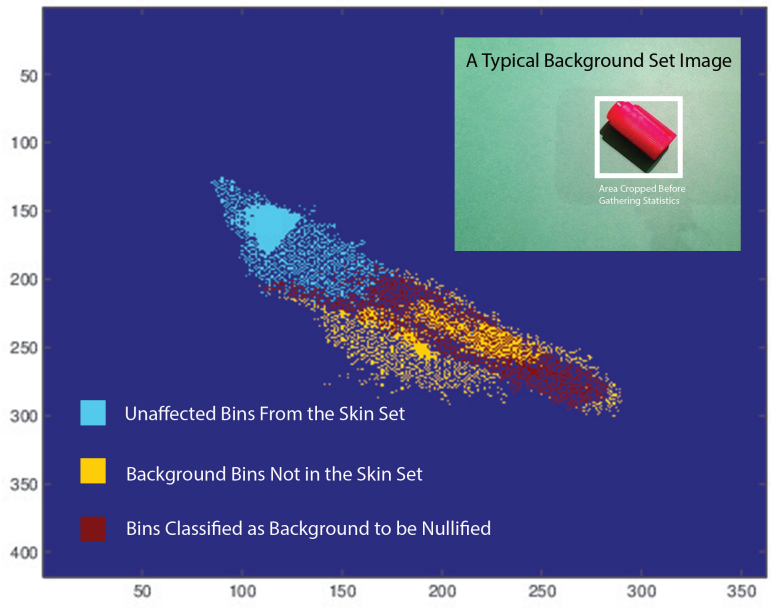


Fig. 3.3 Successful background removal.

3.3.1 RGB Bin Allocation

All the information which is present in the 8-bit unsigned images is in an RGB space. The first problem is that we have a large image set with large images. We're not interested in what these images are pictures of; we're only interested in the individual pixel values themselves. So, we produce a 3D histogram with one bin for each RGB combination. This gives us a histogram with 256x256x256 bins. This is a very large data set, perhaps unnecessarily so, but it's easier to work with than a set of images and is guaranteed to contain all the relevant information for the statistics.

The algorithm is written in MATLAB, and it very simply loads up each image in the set and runs through each pixel, incrementing the corresponding bin. As it's going, it keeps a count of the number of bin allocations, so we have a total pixel count. After it's run through all the images, it find the largest bin and keeps a record of the largest bin count, so we can have a normalized bin when requested.

3.3.2 Skinning the Bins

Next, the bins which are at the extreme edges (i.e. the ones which correspond to the outer faces of the RGB cube) are set equal to 0 with a specified depth. For example: if a depth

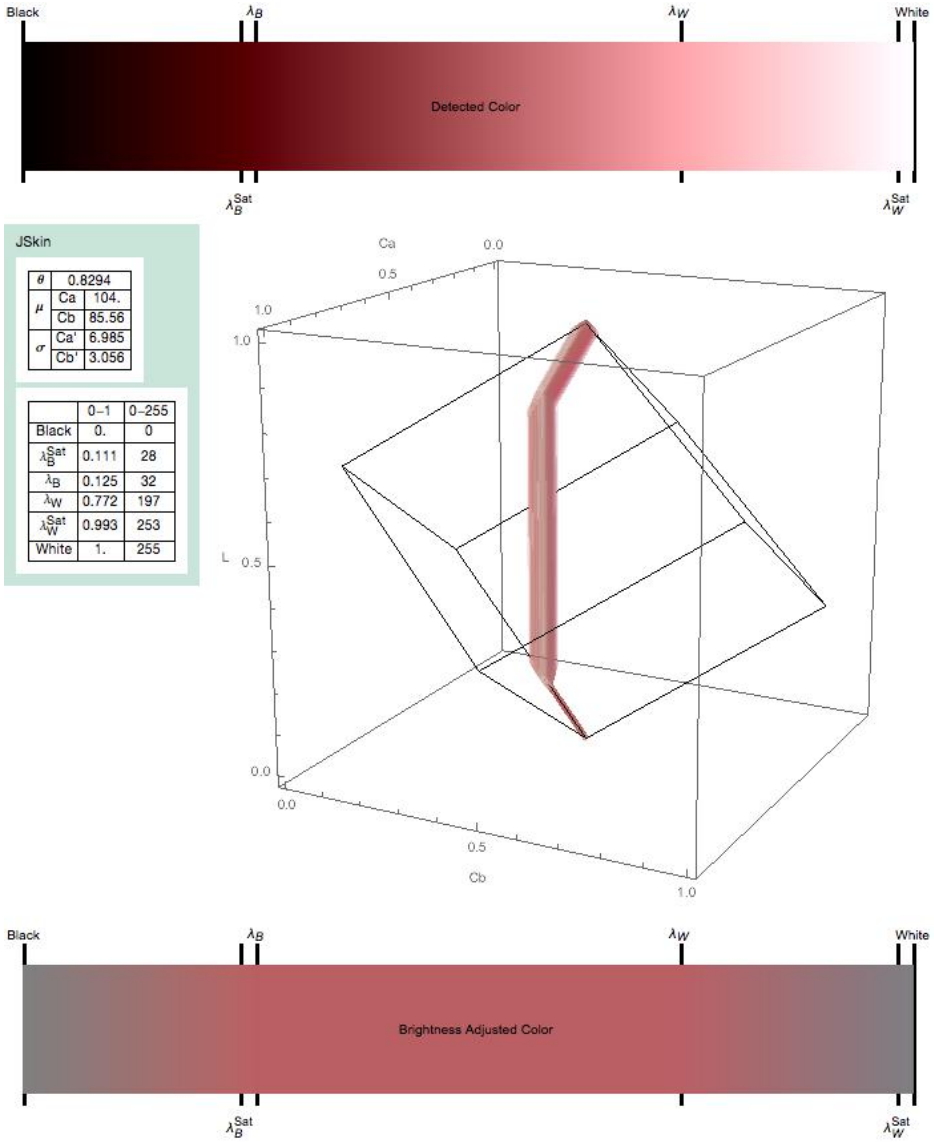


Fig. 3.4 White-out and Black-out.

of 3 is requested, bins of positions 0, 1 and 2, and 253, 254 and 255 are set to 0 in all three dimensions, and the total pixel count is adjusted to compensate for the nulled bin counts. This is done to address problems of white-out and black-out as described in 3.2.3.

It should be noted that the reason the RGB cube is skinned before rotation is because it's not as easy to do in the rotated color space.

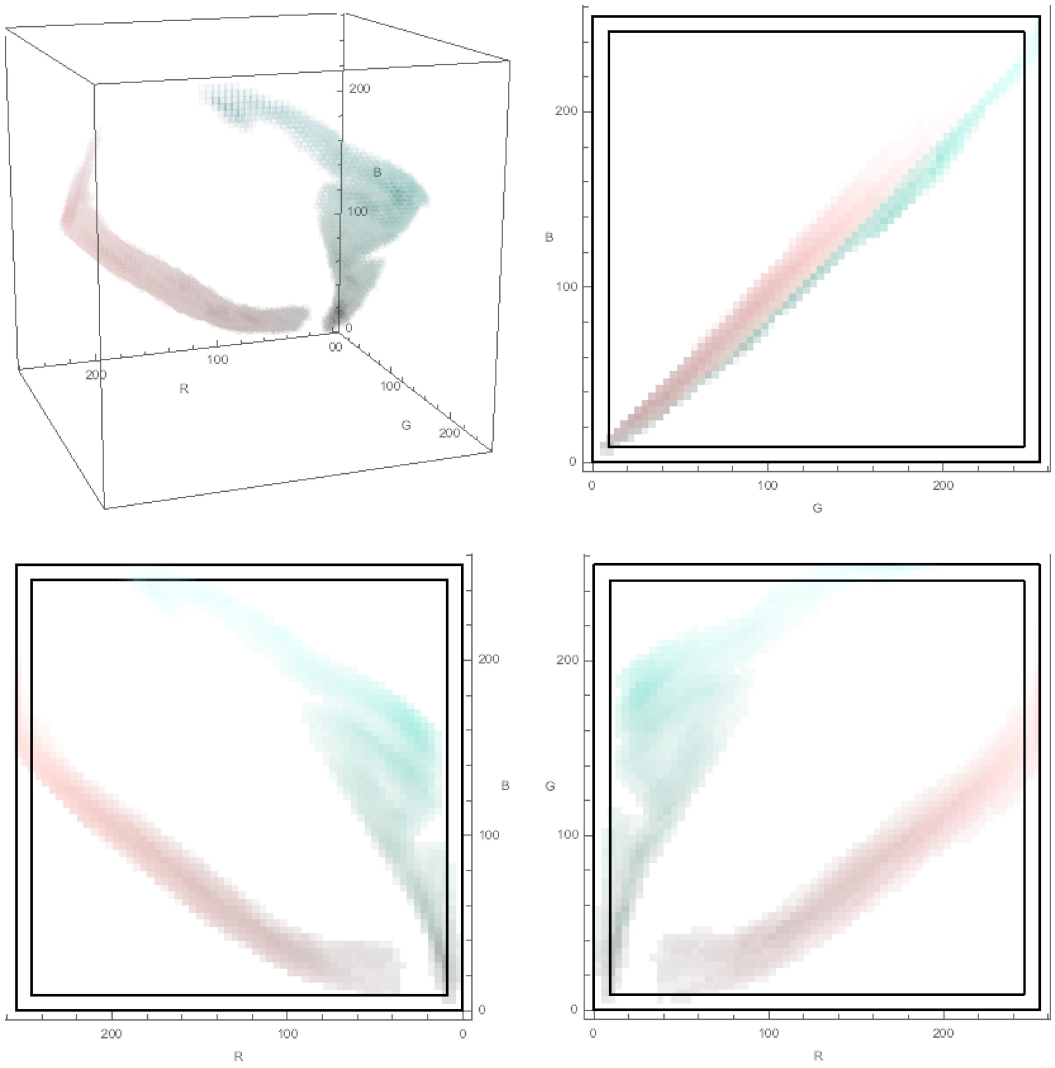


Fig. 3.5 **The 3D RGB histogram**. Distinct distributions can be seen for the Skin and the background. Here the bin color corresponds to the color which that bin represents and the opacity indicates the frequency of that bin value.

3.3.3 Rotating the Bins

Because each of the bins corresponds to just one RGB value, we can find the equivalent bin in the LCaCb space simply by rotating the bin index. This is done using the normalized rotation as described in Chapter 2. The normalised rotation is used as we desire the mean μ and standard deviation σ to be specified in the $0:1$ range, which is easier to find where all the axes are of the same length. With this done, we now have a set of bins in the LCaCb color space equivalent to that which would be found if we had applied the transform to each of the images and then collected the statistics from the transformed images.

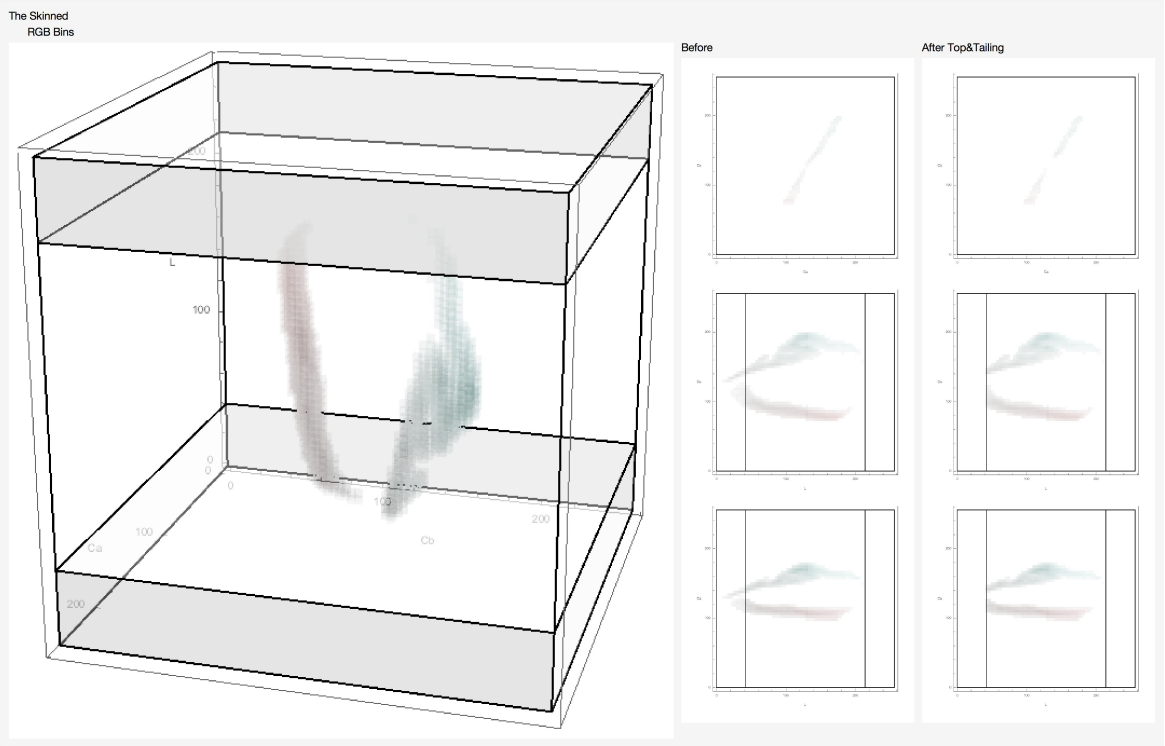


Fig. 3.6 **The 3D RGB histogram after 'skinning'**. Distinct distributions can be seen for the Skin and the background. Here the bin color corresponds to the color which that bin represents and the opacity indicates the frequency of that bin value. Bin counts outside the black lines are set to zero.

3.3.4 Top and Tail

Naively collapsing the bins along the luminosity axis artificially skews the chromatic distribution along the axis which passed through the luminosity axis; this is easily explained due to white-out and black-out (3.2.3). Although we've "skinned" the bins, the white and black tips of the cube suffer from white-out and black-out more than any other regions, and there's a tendency for pixel values to converge under the white point and the black point without necessarily hitting the side of the cube first due to the iPhone contrast and brightness adjustment. This can be seen in Figure 3.8.

We collapse the bins excluding the bins which are clearly suffering from white-out and black-out. This could be done mathematically by taking the bin of the distribution which is furthest from the luminosity axis, and then finding the intersection with the RGB cube when this chromatic value is at its limits, just before it reaches the edge of the cube where it suffers from white-out or black-out. But it's a simple matter to look at the three projections of the 3D LCaCb bins and manually determine the limits for the valid region.

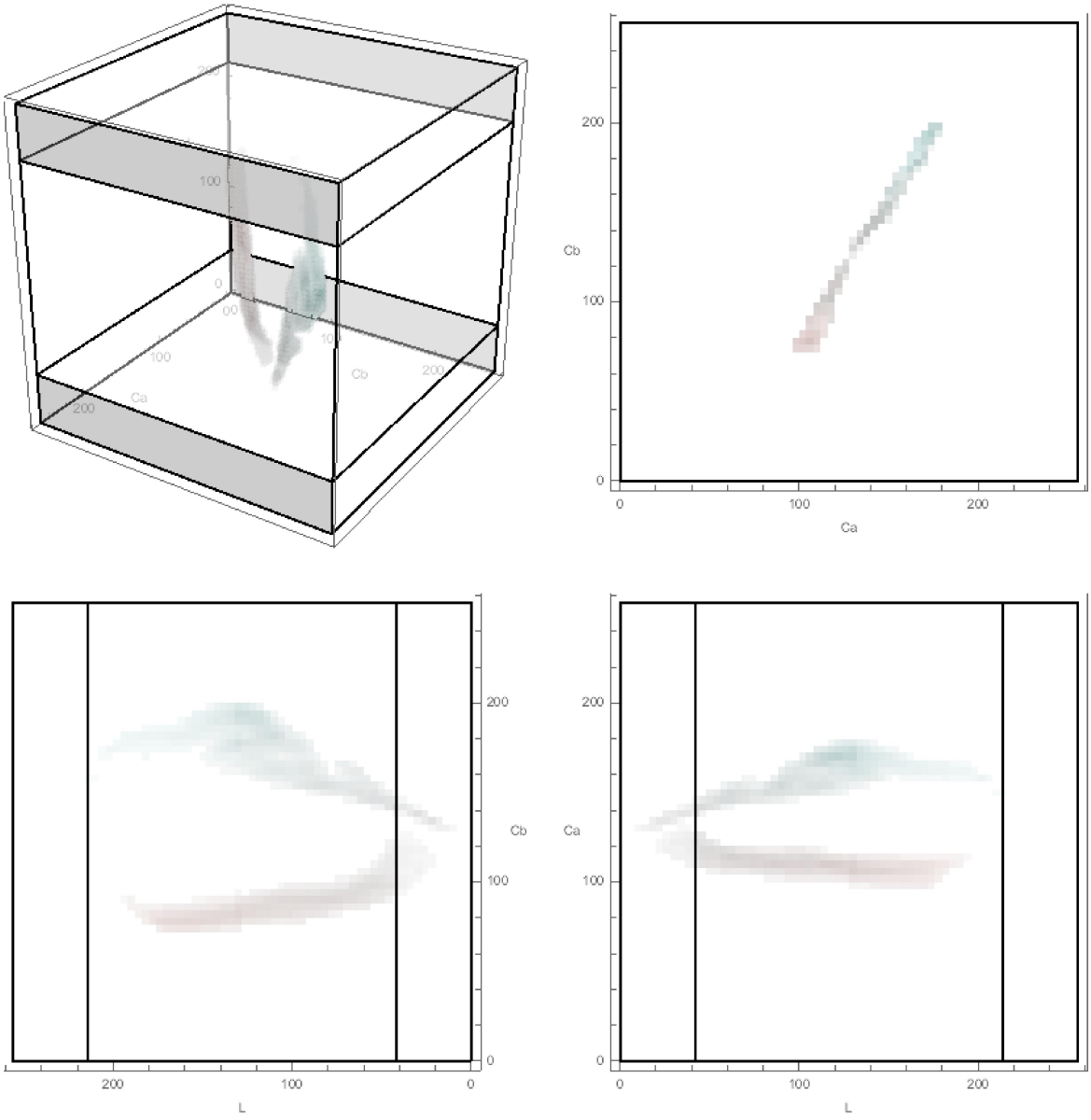


Fig. 3.7 **The 3D RGB histogram after rotation**. The skin and background distributions can be seen for the Skin and the background. Here the bin color corresponds to the color which that bin represents and the opacity indicates the frequency of that bin value. Bin counts outside the black lines are set to zero.

3.3.5 Collapsing the Bins

Because we're modelling the chromatic space and not the luminosity, we now collapse the 3D histogram by summing the bin values along the luminosity axis. So, we now have a 2D histogram in CaCb chromatic space.

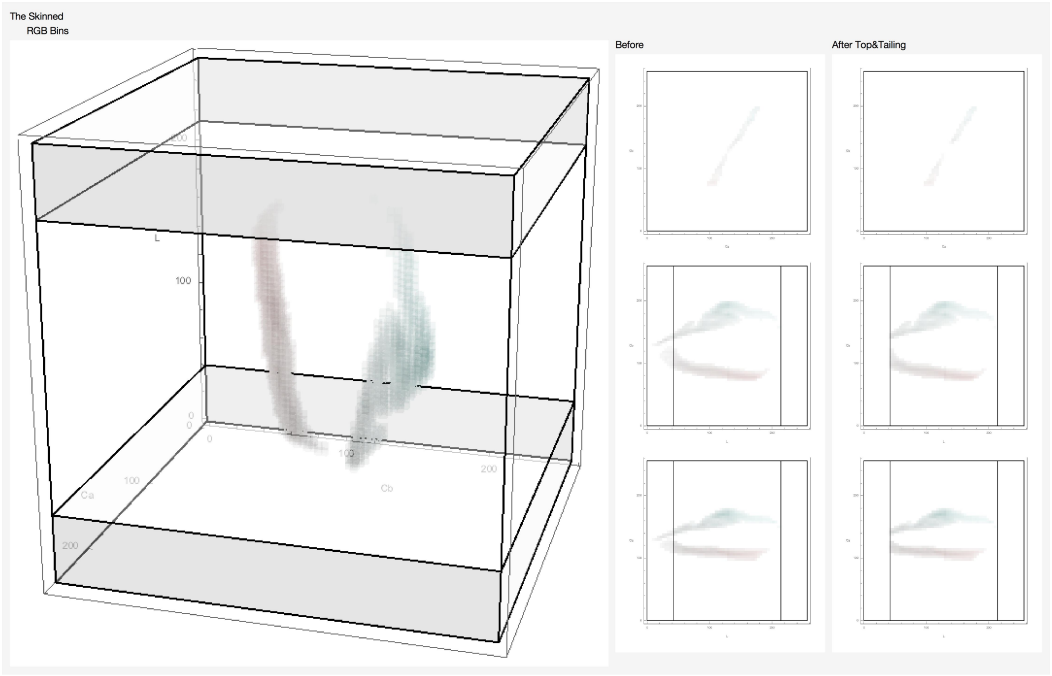


Fig. 3.8 Filling the RGB Bins

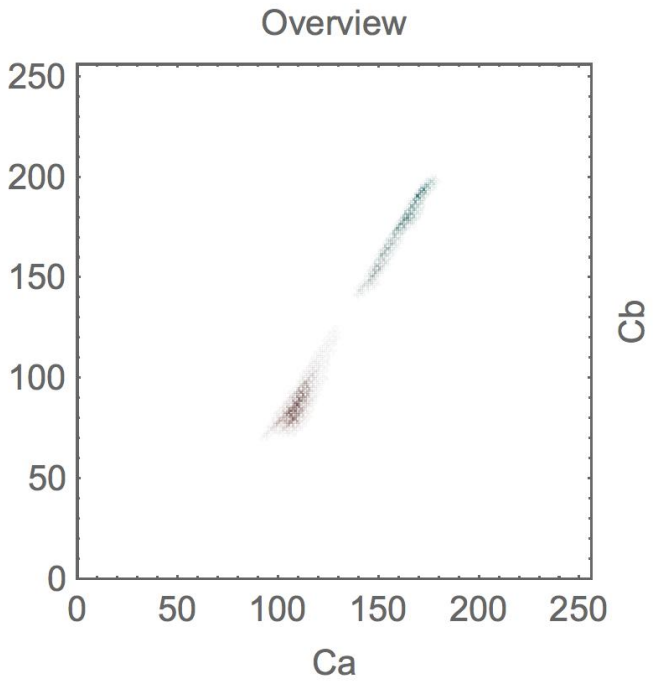


Fig. 3.9 **The 2D histogram after summing along the luminosity axis**. The color corresponds to the chromatic value of the bin at average luminosity. The opacity corresponds to the frequency of the chromatic value.

3.3.6 De-Speckling the Bin Values

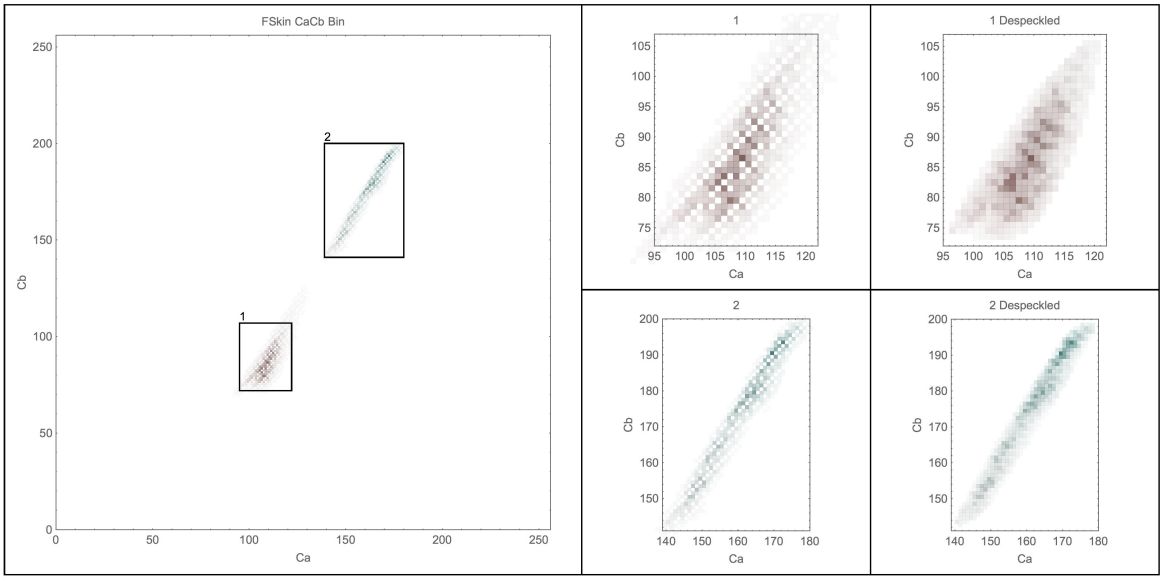


Fig. 3.10 **The De-speckled Bins**. The action of the de-speckling algorithm can be seen in the right-hand close-up panels

The raw camera output has, by this stage, undergone two rounds of processing — first in the device, before the AP layer, creating an 8-bit RGB image; and then in the rotation to the LCaCb space. These processes are, unfortunately, not 1-to-1. This results in some bins being artificially overpopulated and other bins becoming artificially empty. The effect of the LCaCb rotation can be controlled by extending the axis lengths; this process can make for at worst 1-to-1 correspondence. However, this necessarily introduces a greater number of inaccessible bins. In terms of collecting the statistics, these effects are not problematic aside from the fact that it introduces empty bins inside the main region of interest, which causes difficulties for the algorithm further down the line. Graphically, this problem looks like speckling. This speckling is also apparent in the RGB bins, and it is assumed this is a result of the pre-processing of the image by the device before the AP layer. It is noteworthy that although the camera claims to capture full 3-channel, 8-bit RGB information, this is not quite true.

In attempting to solve this problem, the obvious idea is to find all the non-zero bins and fit an interpolating function between them. This will effectively remove the empty bin artefacts within the densely-packed region which is the distribution we're interested in. However, the empty bins outside the main distribution are not artefacts and are genuinely empty bins, so removing all the empty bins and then fitting the function joins together any

outlying points or secondary distributions corresponding to regions such as the background. We therefore desire a method which will allow us to keep all the non-empty bins and the empty bins outside the main distribution, i.e. all the genuinely empty bins. To achieve this, we designed a Matlab routine which essentially paints a region around each non-zero point, marking it as part of the main distribution. It then takes all the unmarked regions and includes all the empty bins in those regions. So, the set of points which is all the non-empty bins and all the empty bins in the unmarked regions satisfies the requirement, and a simple interpolating function can easily be fitted to those points.

3.3.7 Blob Detection

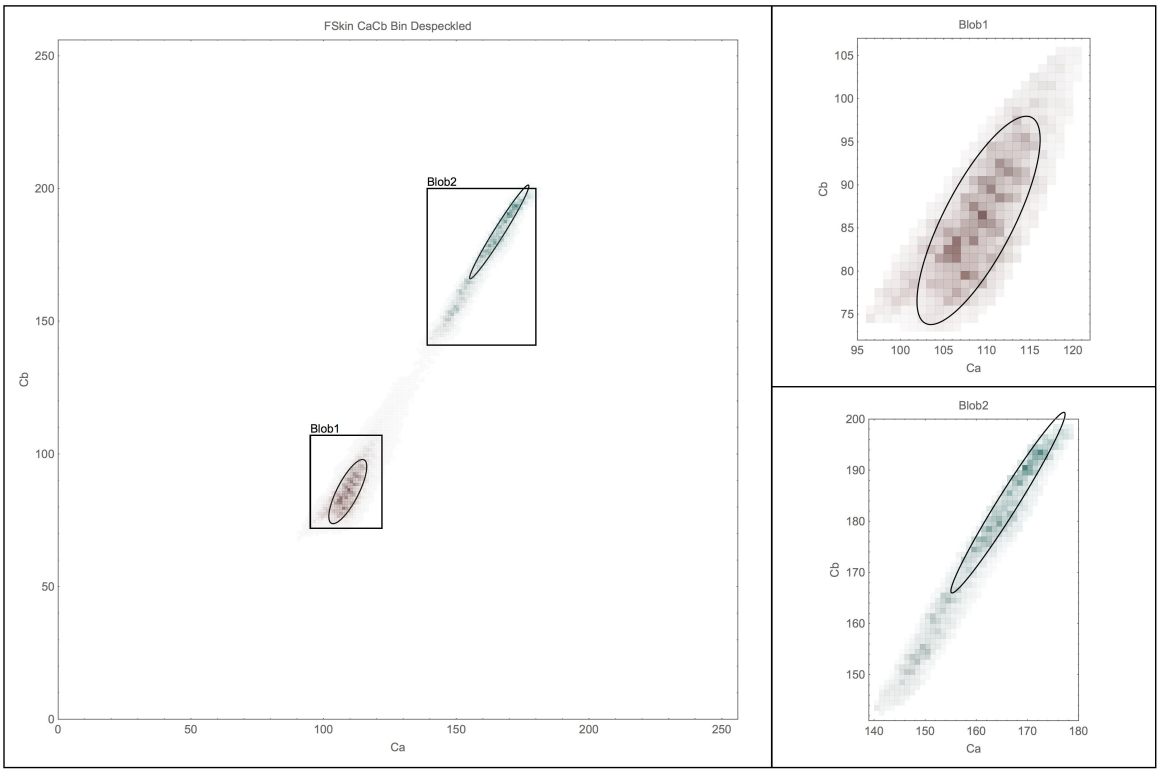


Fig. 3.11 **The Blob Detection.** The detected blobs can be seen in the right-hand panel with the ellipse fit overlayed. The split does not depend on the ellipse but on the extreme edges of the blob.

Having removed the empty bin artefacts and compensated for white-out/black-out effects, the final step is to remove the bin counts of the bins which are associated with the background, thereby leaving a distribution which corresponds to chromatic skin values and which is artefact and systematic-error-free. With the chromatic bins processed as they have

been so far, it is clear that there is a distinct distribution for the skin and a distinct distribution for the background. However, a distribution for the background was produced earlier, compensating for the iPhone's pre-AP-layer processing.

Using MATLAB's blob detection algorithm, we can find the distinct patches of chromatic information. We expect there to be two distinct blobs: one corresponding to the target skin values, and one to the monochromatic background. The distribution is divided into two, one for each of the detected blobs (Figure 3.11). The blob detection algorithm also returns the center $\widetilde{\mu}$, eccentricity $\widetilde{\theta}$ and major and minor axis $\widetilde{\sigma}$ for an ellipse which most closely fits the blob shape. The distribution which contains the blob center closest to a reference skin value is chosen to be the skin distribution and the ellipse values are passed to the next step where the Gaussian fit is obtained. Only the bin values between the extreme edges of the blob are passed on to the next stage.

3.3.8 The Gaussian Fit

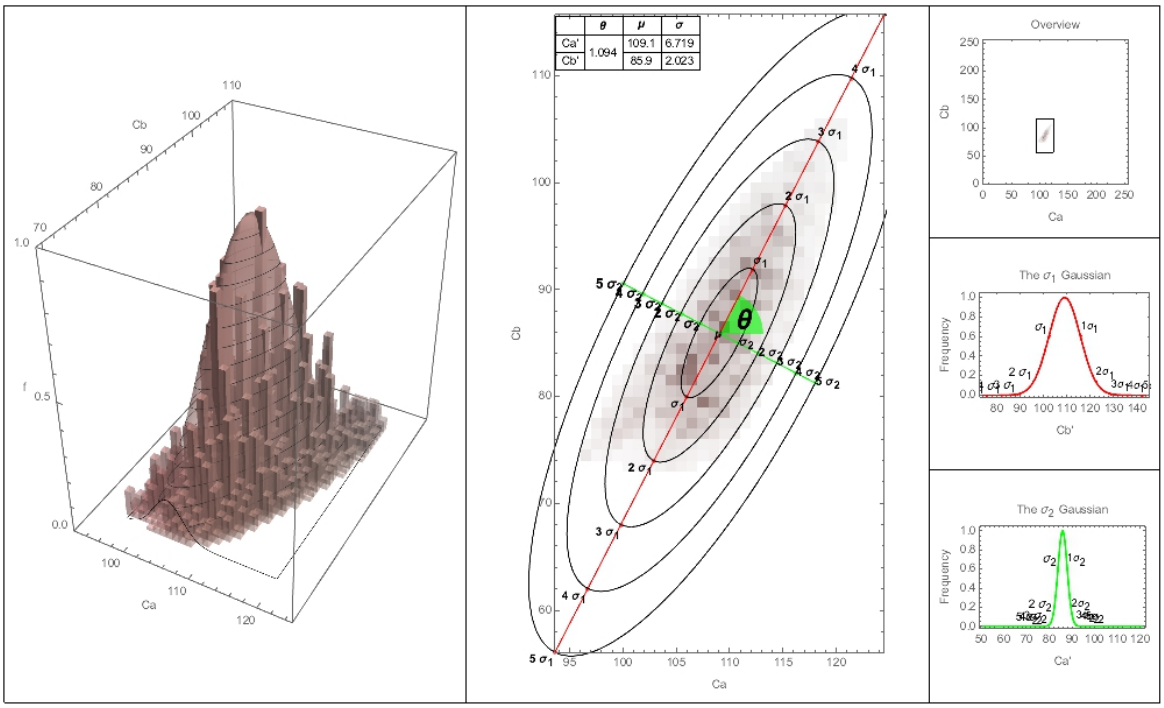


Fig. 3.12 The Gaussian fit to the blob

We now have a histogram which has bin values relevant to the target and no others. We now fit a 2D Gaussian to a normalized histogram values, initializing the routine with the values, the blob center $\widetilde{\mu}$ for the mean μ and the elliptical axis lengths $\widetilde{\sigma}$ for the standard

deviation σ and the eccentricity $\widetilde{\theta}$ for the orientation θ, provided by the blob detection.

A least squares fit of the 2D Histogram with a 2D gaussian function (eq. 3.1) is found using MatLab's lsqcurvefit.

$$\textbf{Let} \quad \overline{\textbf{Ca}} = \mathrm{Ca} - \mu_1 \quad \text{and} \quad \overline{\textbf{Cb}} = \mathrm{Cb} - \mu_2 \tag{3.1}$$

$$\exp\left(-\frac{\left(-\overline{\textbf{Ca}}\,\sin(\theta)+\overline{\textbf{Cb}}\,\cos(\theta)\right)^2}{2\sigma_2^2}-\frac{\left(\overline{\textbf{Ca}}\,\cos(\theta)+\overline{\textbf{Cb}}\,\sin(\theta)\right)^2}{2\sigma_1^2}\right) \tag{3.2}$$

3.4 Sample Sets and Results

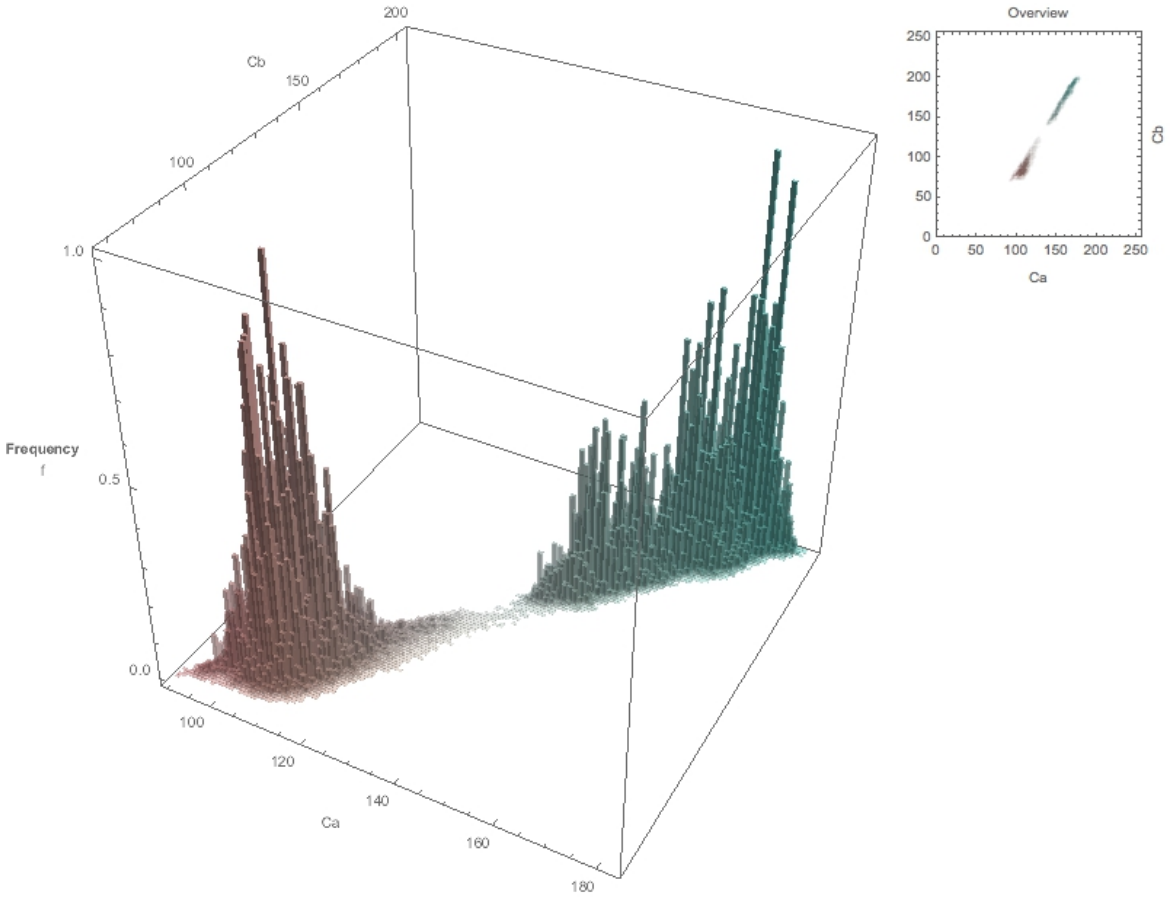


Fig. 3.13 The Ca Cb histogram for the combined F,J&N sets incliding the background. The colors indicate the pixel color corresponding to the bin.

Sample image sets were collected using the iPhone for three individuals with varying skin tones. Sets of images were taken for each digit and the hand as a whole under different lighting conditions and orientations. A selection of the images can be seen in Figure 3.16

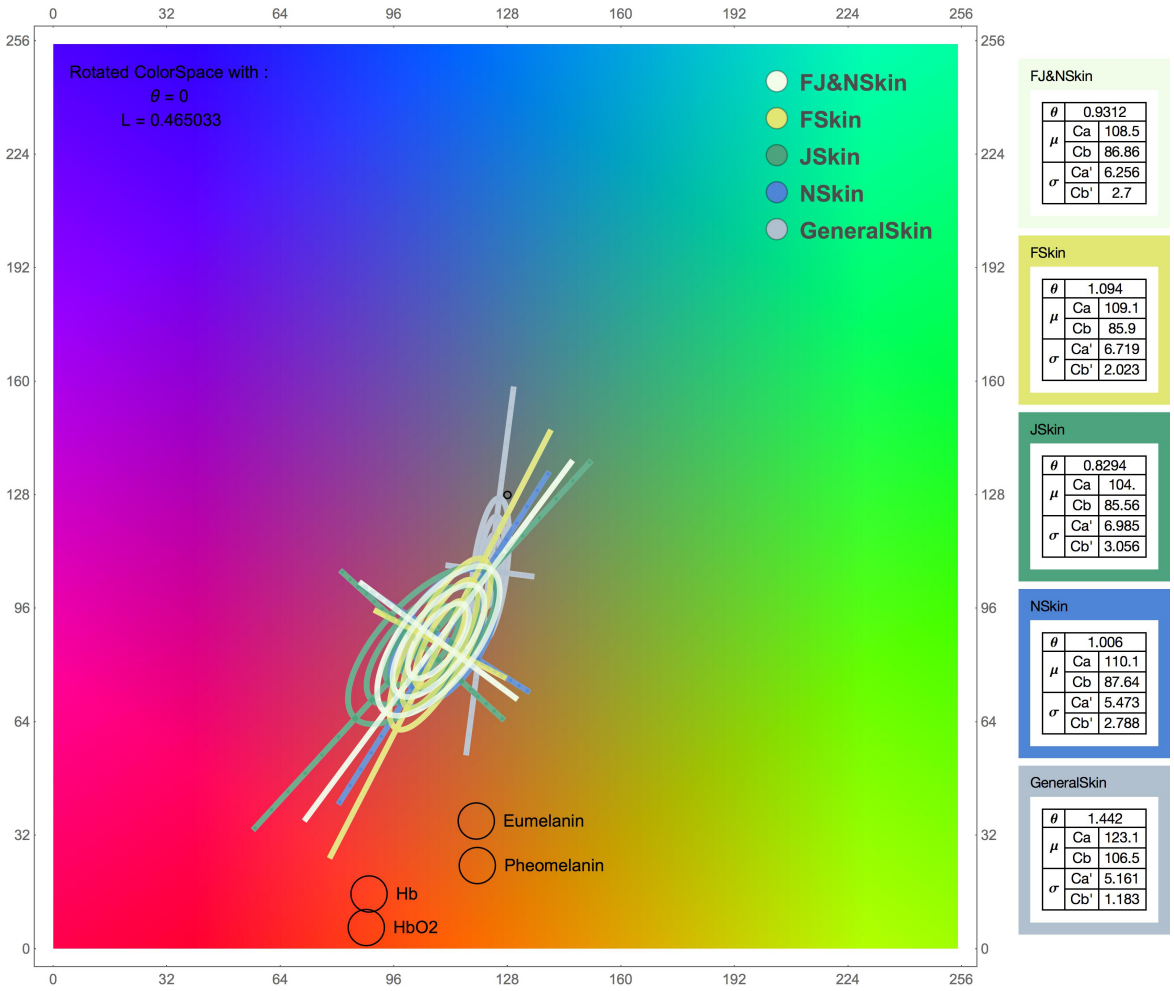


Fig. 3.14 The four sets and the combined set of individuals. The ellipses are placed at 2σ, 3σ and 4σ positions. Values inside 2σ would be kept by the distribution function.

The MATLAB bin class was written so that the combined statistics for the three individuals could be found by adding the RGB bins together, and then following the same steps 1-8 as described previously. All three distributions along with the combined distribution can be seen in Figures 3.14 and 3.15.

In an attempt to identify a larger range of values for the skin space, a large number of skin samples were taken from photographs from the Humane project by Angelica Dass, an ongoing "chromatic inventory" art project which aims to compile every possible human skin color, categorized by the PANTONE guide color classification system (**?**). The skin colors catalogued thus far are independent of race or ethnicity, so the samples are representative of human skin tones in general.

The image set consists of approximately 200 skin samples each showing only one indi-

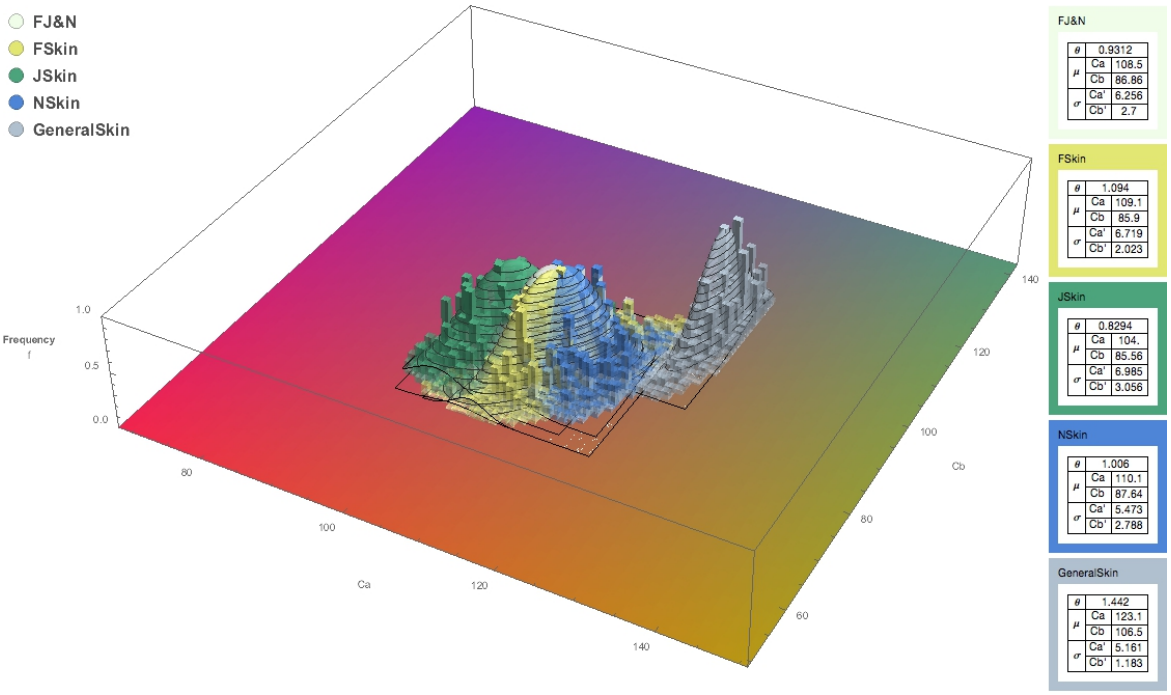


Fig. 3.15 The histograms for the four data sets and the combined set of individuals are shown alongwith the Gaussian fits to the histogram. The color indicates the data set.

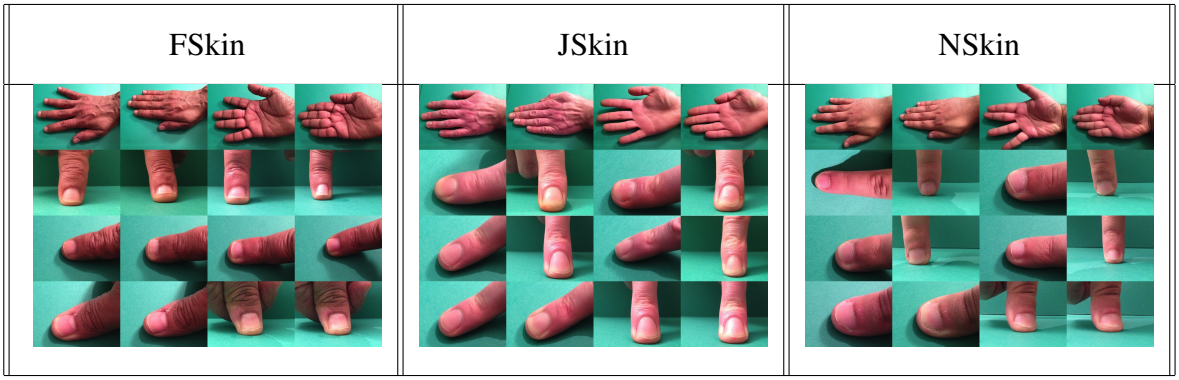


Fig. 3.16 Samples from the image sets for each individual.

viduals skin. The result can be seen in Figures 3.14 and 3.15 as the GeneralSkin set. The distribution stands apart from the others and surprisingly has a smaller standard deviation. This is probably due to camera differences and the sampling techniques used in the Humane project which focus on finding an average color rather than the channel values in camera. The orientation and position of the distribution is more consistent with the theoretical values for the skin pigments than those for the individual sets. This is likely due to the fact that the theoretical values were calculated for a different camera CCD, the iPhone CCD characteristics not being available. For this reason the general distribution is excluded from

further consideration aside from the recognition that different cameras will require bespoke statistics to be gathered.

In contrast to the GeneralSkin set the individual sets are close together with similar standard deviations however the angle varies significantly. The angle needs to be fixed so that the statistics can be gathered in the rotated color space. The angles are summarised in the table below:

Set	Decimal	Approx	Relative
FJ&NSkin θ_{FJN}	0.9312	$\frac{75\pi}{253}$	θ_{FJN}
FSkin θ_F	1.0940	$\frac{39\pi}{112}$	$\theta_{\text{FJN}} + \frac{3\pi}{58}$
JSkin θ_J	0.8294	$\frac{33\pi}{125}$	$\theta_{\text{FJN}} - \frac{\pi}{31}$
NSkin θ_N	1.0060	$\frac{49\pi}{153}$	$\theta_{\text{FJN}} + \frac{\pi}{42}$

This suggests that taking θ' to be in the range $\theta_{\text{FJN}} - \frac{\pi}{84} < \theta' < \theta_{\text{FJN}} + \frac{\pi}{84}$ is safe as it is at most half way to the nearest individual and the position of the mean remains within 2 standard deviations $\begin{pmatrix} 84 \\ 117 \end{pmatrix} < \mu' < \begin{pmatrix} 83 \\ 121 \end{pmatrix}$. This is somewhat arbitrary and we could extend the region if we had reason but this range is sufficient to ensure that a computationally advantageous value will be within the range being 1/7 th of a Pi/6 region. The perturbation to the channels for the range of possible θ' values can be seen in Figures 3.17 and 3.18.

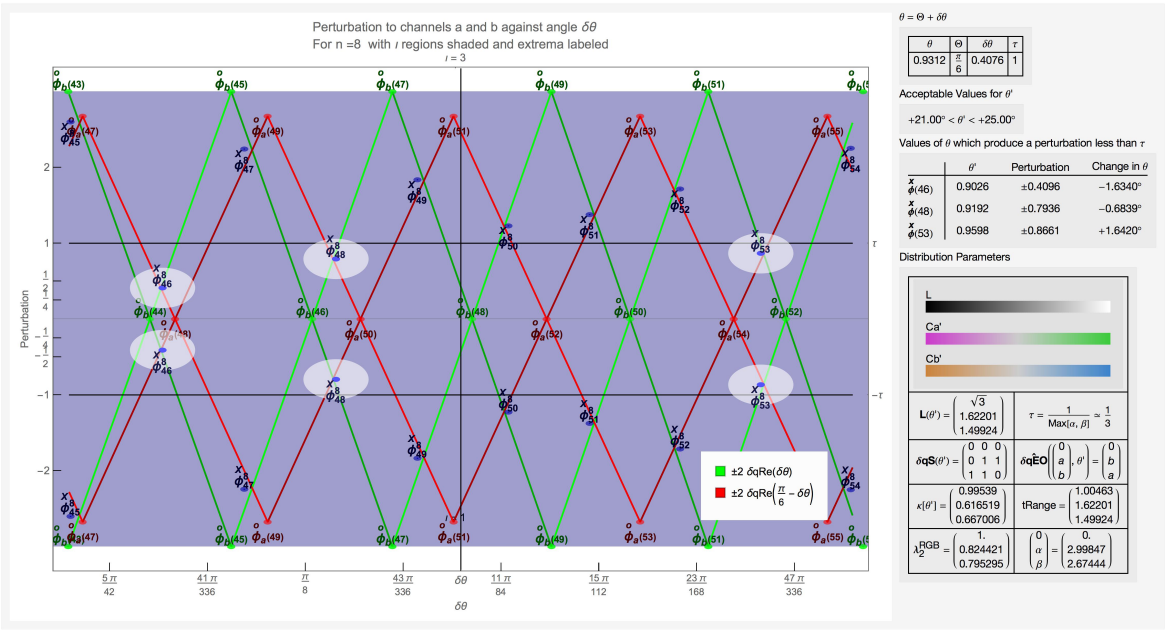


Fig. 3.17 The channel perturbations within $\frac{\pi}{84}$ of θ'. The channel perturbations are scaled by α and β so a tolerance $\tau = 1$ is used. Three angles satisfy the requirements

We now follow the procedure outlined in chapter 2 and obtain an angle informed by

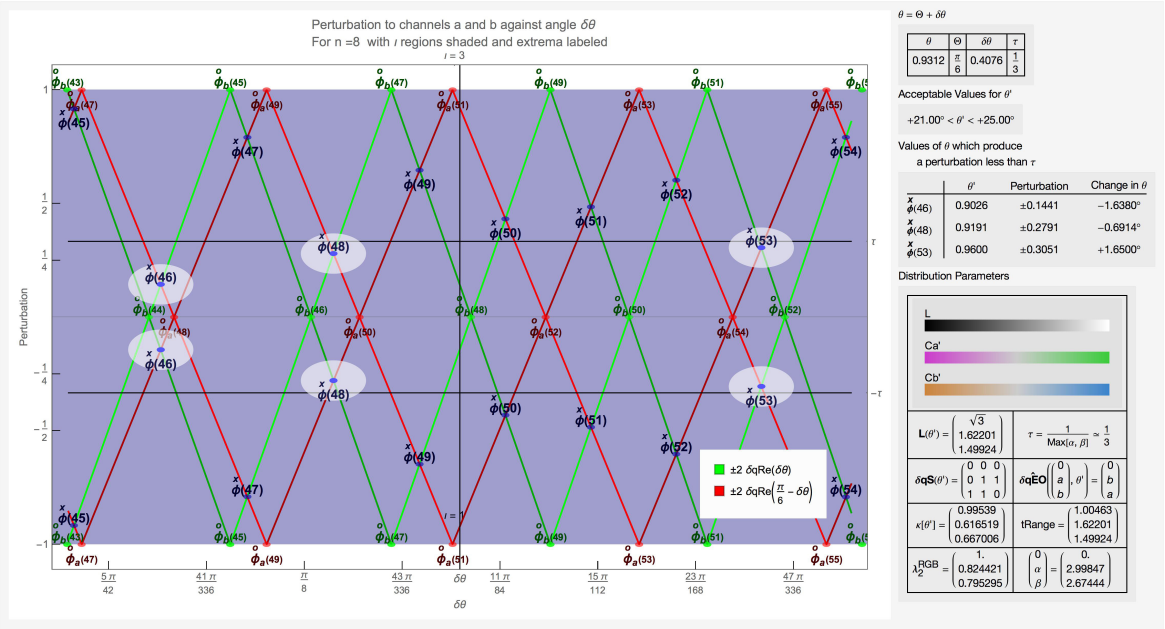


Fig. 3.18 The channel perturbations within $\frac{\pi}{84}$ of θ'.

computational considerations. With the perturbations to each channel scaled by their relative importance (Figure 3.17) there are three angles which satisfy the criteria. The algorithm returns the closest angle to the requested angle which is $\overset{x}{\phi}(48)$ however the perturbation is close to the tolerance of 1 which suggests that if the distribution parameters were adjusted to fit a particular individual then the perturbation may slip above the tolerance. The channel scaling $\{\alpha,\beta\}$ are also fairly close in value so the unscaled perturbations can be used with a tolerance $\tau = \text{Max}(\alpha,\beta)$ Figure 3.18. Accounting for both low perturbation and future adaptability a final angle $\theta = \overset{x}{\phi}(46) = 0.9025768293268257$ was chosen.

As a final step, the MATLAB statistics gathering routine was run again using the combined F,J&N RGB bins as a starting point and then rotating to the LCa'Cb' color space with the new value for θ. This was done partly to check that the code works as expected and to see how well the product of Gaussians function fits the distribution. The only difference to the MATLAB code was that for this run the Gaussian fit was found using equation 3.3 which is the product of two 1D Gaussians with a fixed amplitude of 1. The result can be seen in Figure 3.19.

$$\exp\left(-\frac{\overline{\mathbf{Ca}}^2}{2\sigma_1^2}-\frac{\overline{\mathbf{Cb}}^2}{2\sigma_2^2}\right) \quad \textbf{where} \quad \overline{\mathbf{Ca}} = \text{Ca}-\mu_1 \quad \text{and} \quad \overline{\mathbf{Cb}} = \text{Cb}-\mu_2 \tag{3.3}$$

It can be seen that the least square fit is a good predictor for the distribution but the aim

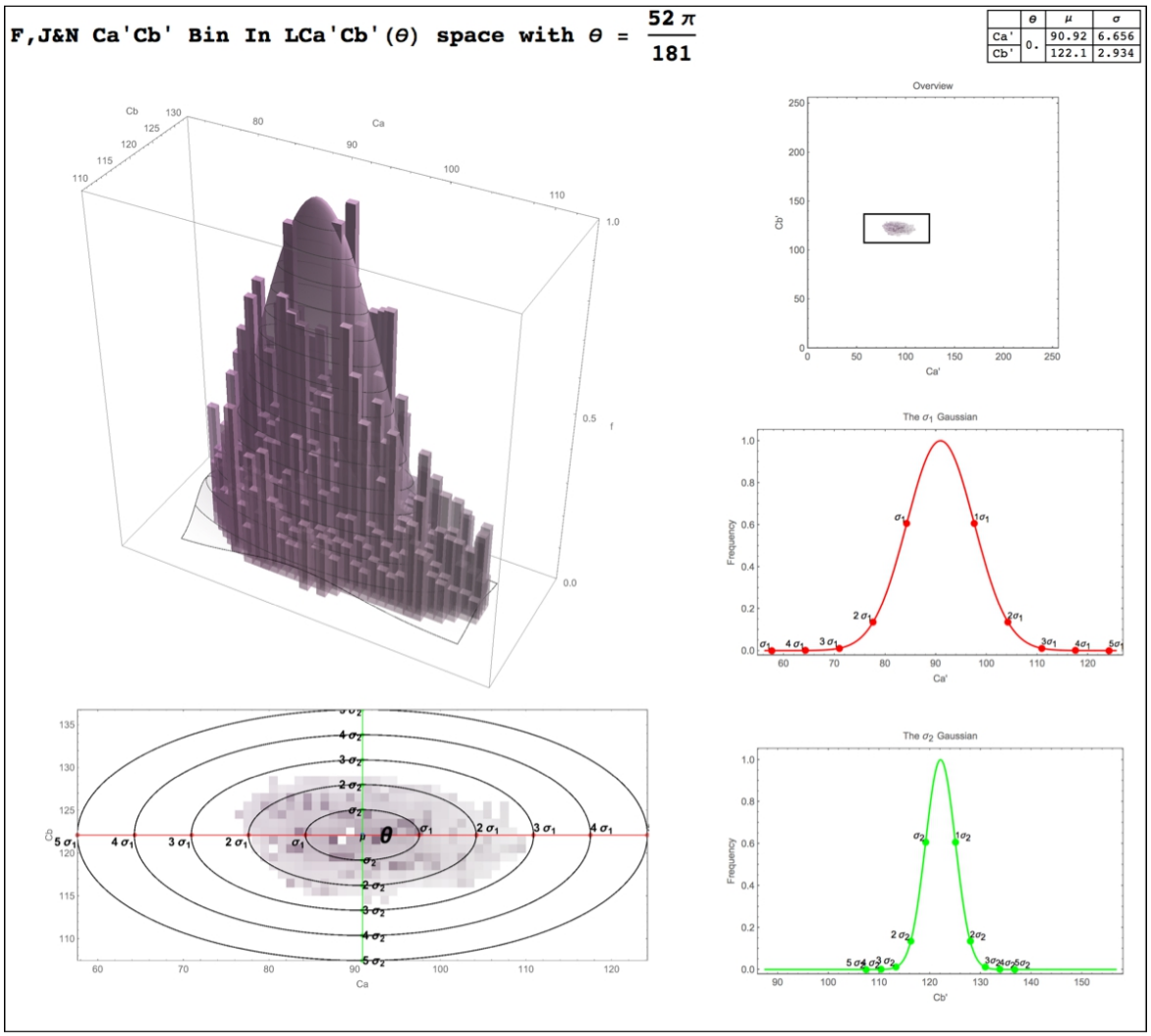


Fig. 3.19 The Product of Gaussians fit to the three individuals statistics in the LCa'Cb' color space.

of the color space is to capture all skin relevant chromatic information. For this reason, as well as the desire to be inclusive of skin tones outside of the sample set of three individuals, a region larger than the 2σ ellipse region (Figure 3.21) which is kept by the tight MATLAB fit may be desirable to be included. The 3σ ellipse region would include everything in the combined individual F,J&N histogram. A 5σ ellipse region would likely account for a wide variety of skin tones.

Extending the distribution may also affect the accuracy required in the rotation as mentioned previously. Extending out to a 5σ ellipse region does indeed affect the tolerance (Figure 3.22), but the chosen angle perturbation ± 0.144 still satisfies the tolerance $\tau = 0.3094$ as $0.1441 < 0.3094$. It's also notable that the tight MATLAB distribution requires a partition in

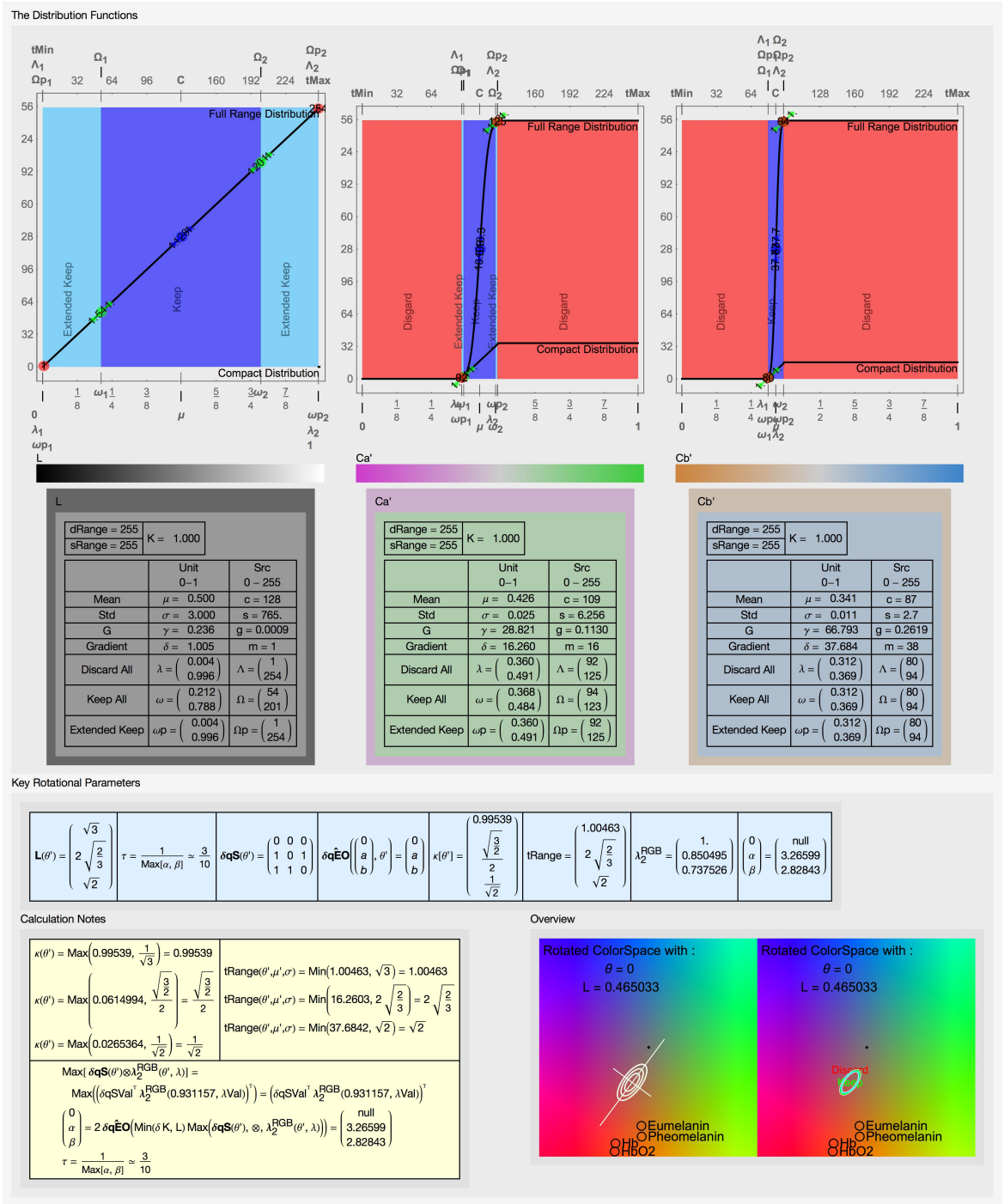


Fig. 3.20 The redistribution functions for each axis alongside summary tables of the key parameters and a selection of calculations and an overview of the distribution in the new color space

the chromatic axis while the extended distribution requires a small amount of redistribution. The tight MATLAB distribution and the extended distribution can be seen in Figure 3.23.

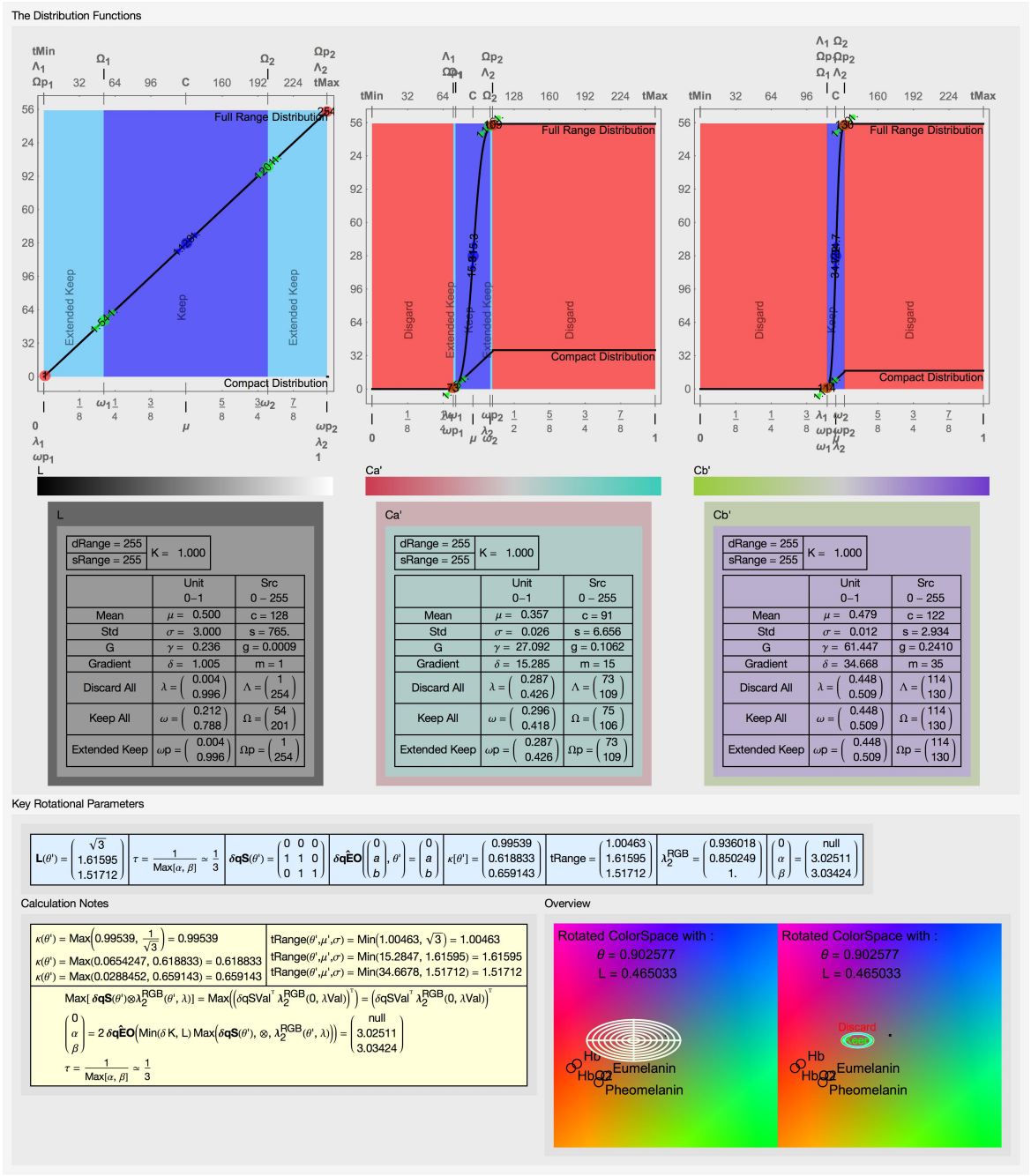


Fig. 3.21 **The Tight MATLAB fit in LCa'Cb'.** The redistribution functions for each axis alongside summary tables of the key parameters and a selection of calculations and an overview of the distribution in the new color space.

3.4.1 Conclusion

Our goal was to find appropriate values for the 2D chromatic Gaussians in order to facilitate the image processing, and to create a color space that preserves all of the chromatic skin

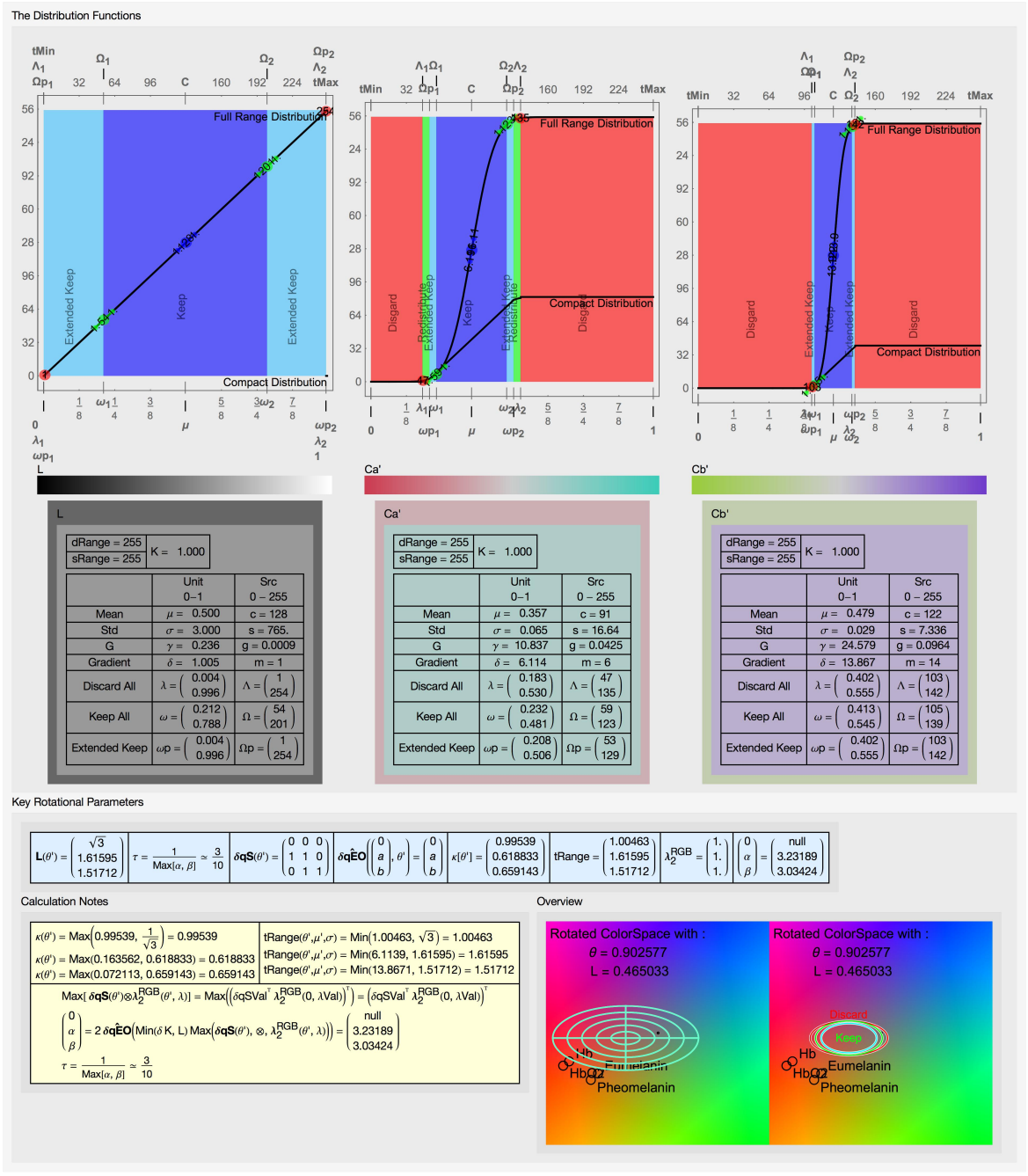


Fig. 3.22 **The extended 5σ ellipse region MATLAB fit in LCa'Cb'.** The redistribution functions for each axis alongside summary tables of the key parameters and a selection of calculations and an overview of the distribution in the new color space.

information. To show that we have achieved this, we will quickly demonstrate the skin probability map:

After redistribution we have two new spaces in which to express the stats parameters: the

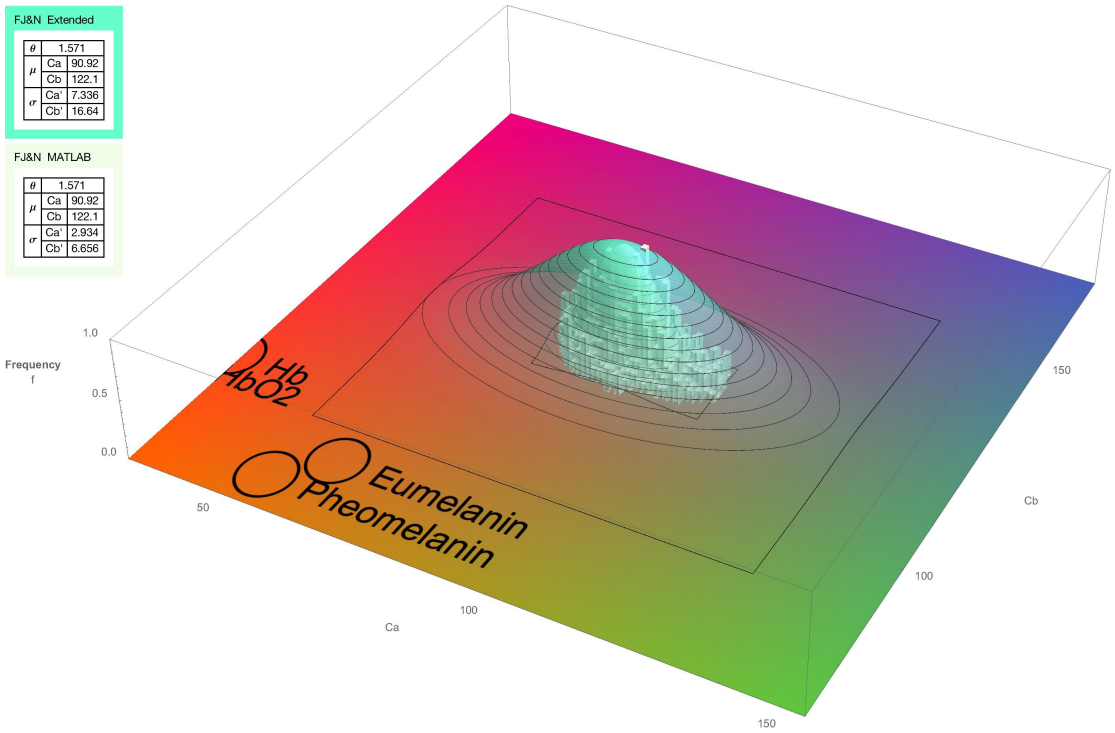


Fig. 3.23 The tight and extended fits in the LCa'Cb' color space.

0-dstMax range and the new 0-1 range, where 0 and 1 correspond to different positions in the original unit space because of the compression of the information. The mean values are at the halfway point in both $c_d = dstMax/2$ and $\mu_d = \frac{1}{2}$. The standard deviations are scaled to fit the region which is 'kept' $\sigma_d = \frac{\sigma}{\lambda_2 - \lambda_1}$. The probability of a pixel in the redistributed color space can then be found using

$$P(L,Ca,Cb) = Exp(-\frac{Ca-\mu_d^{Ca}}{2\sigma_d^{Ca}} - \frac{Cb-\mu_d^{Cb}}{2\sigma_d^{Cb}}) \tag{3.4}$$

the result can be seen in Figure 3.24.

The probability in these images is indicated by the opacity, but as you can see from the above images, all of the skin detail is preserved, therefore all the skin information is preserved. In a normal color space, the preservation of skin information is computationally intensive, but in the case of our bespoke color space, it is simply the product of two 2D Gaussians, therefore finding the skin probability is a very simple function. Thus, we have not only preserved the skin information, we have produced a simplified version of a complicated computer vision task.

It should be noted that the strict 1σ result misses some of the skin; this was expected as

Fig. 3.24 Images of hands from the three individual sets processed with the opacity given by the probability that the pixel value is skin using the simply product of Gaussians method given above.

some histogram values lie outside the Gaussian fit, so loosening the criteria is necessary to capture all of the skin information. A 2σ or 3σ criteria should be sufficient to capture all of the skin information. Additionally, the simple implementation here does not account for unreliable white-out and black-out values, or use nearest pixel values, but this is not the final step of the algorithm. For this, we need a classifier, which will be explored in the following chapter.

Chapter 4

Pattern Recognition and Implementation

4.1 Implementing the Color Space

In the previous two chapters, we have been approaching the color space design problem mathematically. Here, we will outline the practical implementation of the color space algorithm on the mobile device.

The first and most pressing consideration is limited processing power; while our mathematical approach to the algorithm design is significantly faster and more efficient than the more typical color space algorithms — as will be discussed in the following chapter — minimizing the number of clock cycles spent on continuous processes is necessary in order to ensure efficient operation. In the case of redistribution, the integer matrix transform is performed first, which puts the pixel values in the quantized working type *qRsRange* described in Chapter 2, then pixel classification is performed before any of the chromatic information is redistributed as illustrated in Figure D.1.

The pixel classification is performed before the redistribution because it excludes processing pixels which are of no interest based on both chromatic channels, whereas for redistribution one channel may clearly lie outside of the discard region λ while the other could be kept or redistributed. With the pixel classifier, if one channel lies outside the discard region, the other is also of no interest since the pixel value is outside of the partitioning region, thereby avoiding unnecessary processing time spent on redistribution. The classification is straightforward within the limits on the channel values, so it isn't sophisticated like an elliptical partitioning or a probabilistic Gaussian-based thresholding; it is simply a quick way of excluding the majority of the pixel values which aren't of interest.

Once the pixels have been classified, the redistribution is performed — which requires further processing — or we assign a pixel value from a small set of possibilities and skip

onto the next pixel value. The redistribution function is applied to the working type *tRange*, while the comparison and classification is done in *qRsRange*. This is because the information in *tRange* is compressed to avoid wasting space, and as a consequence of this it does not occupy the full data type. *qRsRange*, on the other hand, being the natural range multiplied by the quantization matrix qR, does occupy the full data type. Since the operation is simply a comparison between numbers, it is of no advantage to compress the information so tightly. Additionally, since *qRsRange* is a signed range, one could look at the bit indicating the sign and immediately tell whether or not a given value will be discarded. Once scaled to *tRange*, the redistribution function is applied — which has a source range of *tRange* and a range *dRange* of the destination type — as outlined in Section C in Chapter 2.

It should be noted that the region between the extended keep region ωp and the discard region λ is not especially large; for even a 5σ result, it's only tens of values wide. As such, we use a lookup table for the redistribution in our actual implementation, thus further reducing the processing overhead. The C++ implementation has a lookup table threshold set to 64, a quarter of the source data type range. If the redistribution region is larger than that, the error function approximation from Chapter 2 is used. This is mainly for future-proofing the implementation; given the limited size of the region, we fully expect to use a lookup table for all practical cases.

As another future concession, the facility to add pixel-based functions to the color space transform, such as the skin probability function mentioned at the end of the previous chapter, is allowed. The color space transform is a threaded process, so there is no way of knowing that the last pixel value that was processed by any given thread was an adjacent pixel, as the parallelization is handled by OpenCV's universal threading library implementations. Pixel-based functions, however, are per-pixel processes, and they come out as separate resulting channels, so they can be added to the color space transform without causing any conflicts.

4.2 Fingertip Model

We need an appropriate finger model so we can accurately align points on the fingertip between successive images in the video stream. We need to do this because otherwise, any difference between the frames will be overwhelmed by physical movement of the digit rather than movement of the blood inside the digit. For this reason, we target the most stable portion of the finger: the nail.

A digit has two joints and three segments: the distal, the middle, and the proximal. Each segment is roughly approximated by a rectangle, and the length of each section approxi-

mately following a 2-3-5 ratio, with lengths counting from distal to proximal, and where 1 is the $\frac{3}{4}$ width of the middle segment. Our finger model then comprises of the lengths and widths of each segment, the positions of the joints (i.e. the knuckles) between each segment, the position of the tip, and a position in the model marking the point at which the digit goes out of frame.

Given the width of the middle segment, we can then have an initial guess that the length of the distal segment is 1.5 times the initial width, the middle is 2.25 times that, and the proximal is 3.75 times that. Additionally, the center of the nail feature is not in the center of the distal segment, but a half unit (in our model units) away from the tip. The model can be laid out in units relative to the width of the digit.

4.2.1 The Finger Shape Detection Algorithm

4.2.1.1 Scale Space

The images captured with the iPhone camera are extremely large, and so for the development of the algorithm, we define three scale spaces: one is the original image; two is an image size which represents the tip of the finger regardless of the distance of the camera from the finger, i.e. a scale relative to the width of the finger in the frame of the camera; three is a small image size for tracking the finger where it's freely roaming in the camera's view. We want these images to be formed without the need for resampling. For that reason, we restrict the choice of image sizes to those which are integer factors of the image dimensions. This is done by finding the integer factors of the original image dimensions and then selecting from those factors to scale the image. It should be noted that, to simplify point correspondences, the integer scaling from the small to medium scales is also kept.

4.2.1.2 Finding the Frame Orientation

It is assumed we will not be using severed fingers, and so we know that the finger must exit the camera frame; the first step of the algorithm is then to detect which of the four edges the finger is entering from. This is done simply by finding the longest on-object path running along the frame sides. So, the algorithm starts by looking down the pixels on a frame side; when it locates a high-quaternary value, it uses the Hurdle method to find the longest path in that direction starting from that point. It continues around all four sides; the longest path found by the Hurdle algorithm is assumed to correspond to the finger entering the frame. From this, we set a frame orientation, which is the angle necessary to rotate the image such

that the finger will be entering the frame from the left, and we set a direction vector pointing inwards from the frame edge found.

4.2.1.3 Finding the End of the Finger Shape

Taking the longest Hurdle path on the frame edge found above, we take the middle point on that path and start a 'Run-Reach' algorithm in the direction found previously from that point. The Run-Reach algorithm proceeds as follows:

It finds a Hurdle path in the primary direction, i.e. perpendicularly away from the frame edge. At the end of the Hurdle path, it finds the Hurdle paths in the two perpendicular directions to its primary direction, i.e. parallel to the frame edge. It then finds the midpoint between the ends of the Hurdle path, and then uses that to start a new Hurdle path in its primary direction. The algorithm continues in this way until the end of the finger is found, as seen in Figure 4.1.

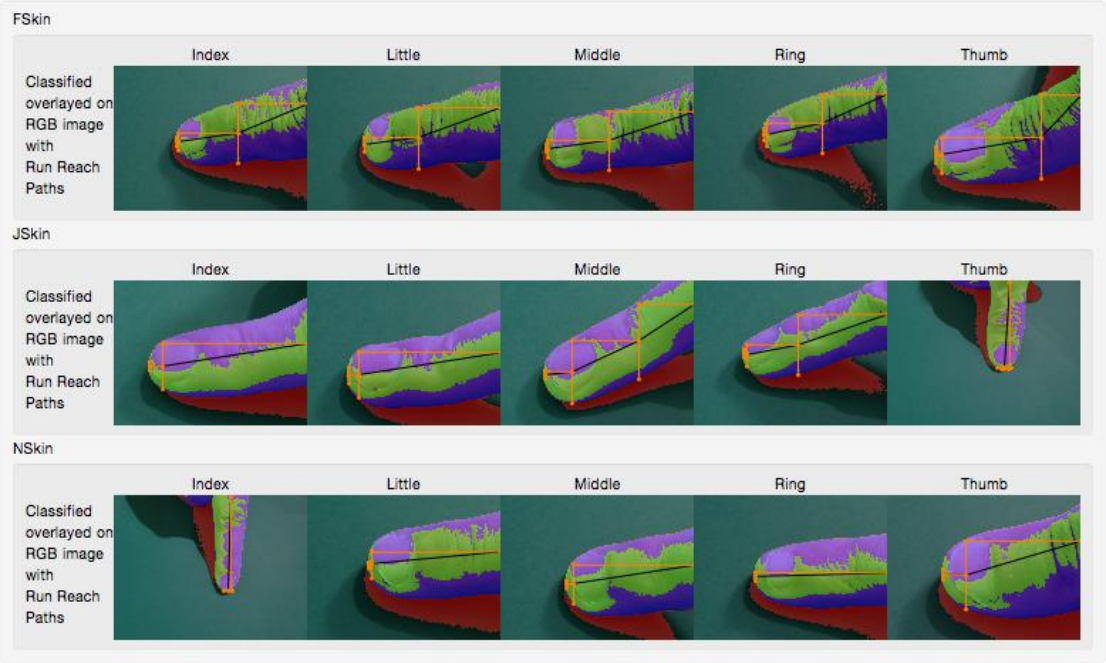


Fig. 4.1 Finding the End of the Finger Shape

4.2.1.4 Filament Fill the Finger

The center points found by the Run-Reach method above form a path on the finger. This is the path given to the Filament Fill method described previously. We now have a set of edge points on the digit. These edge points also have a top and bottom correspondence

which, although they're not perpendicularly-opposite points on the finger, they do allow for a midpoint on the digit to be found. This can be seen in Figure E.1.

4.2.1.5 Exclude Secondary Frame Edge Points and Fingertip Points

We need to exclude from the calculation of the midpoints on the digit values which are near the tip, and values which touch a second frame edge. This is because values toward the tip — given that the finger is not necessarily perfectly horizontally-aligned — give unreliable midpoint values because the Run-Reach path corresponds to shorter and shorter chords on a circle. Values which touch a second edge, more likely than not, do not correspond to the edge of the digit, but are simply where the digit exits the frame (for example, JSkin Middle in Figure 4.2).

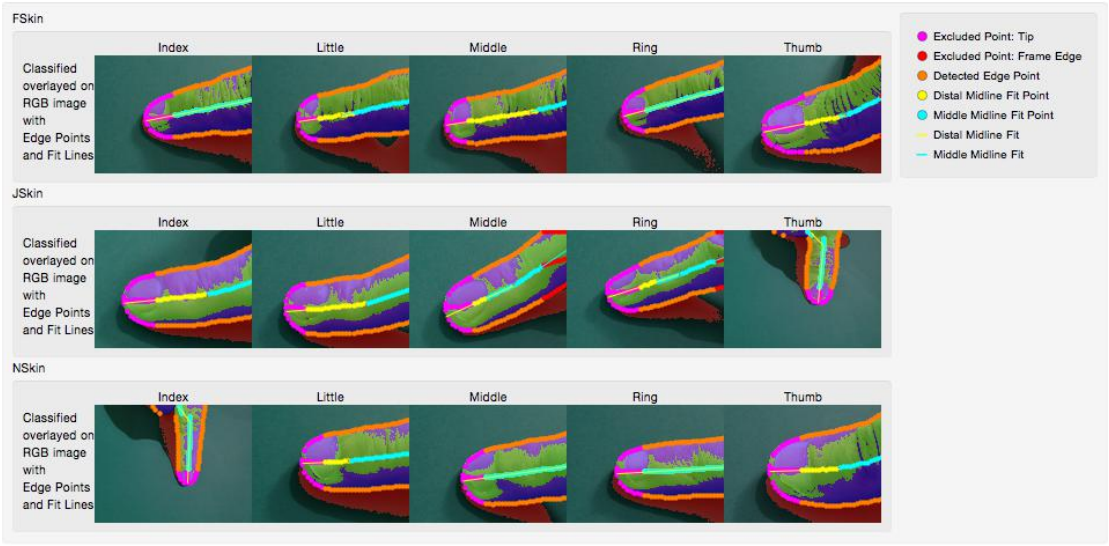


Fig. 4.2 Excluded Secondary Frame Edge Points and Fingertip Points, with the fit to the midline points returned by the Kink Fit algorithm.

To estimate which pairs of edge points correspond to points on the tip, the algorithm finds the length of the Hurdle path for all corresponding bottom-edge to top-edge points. These values are then sorted into ascending order, and then the 'Mean-Median' is taken, ensuring that the average is not affected by extreme values. The tolerance is determined by the algorithm by looking at the difference at the low and high ends of the 66% range used by the Mean-Median.

4.2.1.6 Kink Fit to the Finger

We now have a set of reliable points at the midpoint of the finger. Because the finger is articulated, it is reasonable to assume that it may well be bent. We're interested in the action of the finger being pressed on a surface — as a result, the finger is expected to be articulated at the distal-middle joint, but is otherwise straight. We wish to fit a function which consists of two straight-line segments to the set of reliable middle points. The fit was found using the 'Kink Fit' algorithm described in Section E.0.3.

The Kink Fit algorithm as written takes a set of points where the first component is assumed to be the independent variable and the second component to be the dependent variable. However, because we haven't assumed that the finger comes into the frame from any particular direction, this can be problematic if, say, the finger comes through from the top or bottom of the frame, then the components would be in the wrong order for the Kink Fit algorithm to work. This is easily rectified by simply rotating the points into a standard orientation as if the finger is coming in from the left side of the frame. The computational effort in doing this is minimal because the number of points is small; we're not rotating the entire image, we're merely rotating a set of midline points, performing the Kink Fit algorithm, which returns a set of three points, which can then be rotated back to the original orientation.

In practice, however, the Kink Fit algorithm accepts the full set of midline points and a vector of pointers to the reliable values within that set. This allows the Kink Fit algorithm to extend out the end points of the line to the edges of the digit. Were this not the case, the end points returned would lie somewhere within the digit corresponding with the first and last reliable midline points. The result can be seen in Figure 4.2.

4.2.1.7 Parallel Lines Fit

Our model of a finger assumes that the edges are parallel to each other. Since we have a line which runs through the center of the finger, we can now calculate the distance of the edge points to that line. These distances can be classified as good edge points if they're parallel to the central line. Whether they are parallel is determined by finding the Mean-Median distance from the central line to the edge points using a 50% interval in the distal portion, because it is expected that up to half the points may be in the distal segment, maybe on the tip, and a 75% confidence interval in the middle segment, because we expect fewer than $\frac{1}{4}$ of the edge points to be anomalies. The points are then classified as good if they are of Mean-Median distance from the central line. It should be noted that we consider both the top

and bottom edges when calculating the Mean-Median distance. This technique successfully removes anomalies which are not parallel to the central line.

There is one final step: the points which are classified as good edge points are used to find a linear fit which is parallel to the central line. Finally, the width of the distal and the middle segments is calculated by finding the distance between these parallel lines. This can be seen in Figure 4.3.

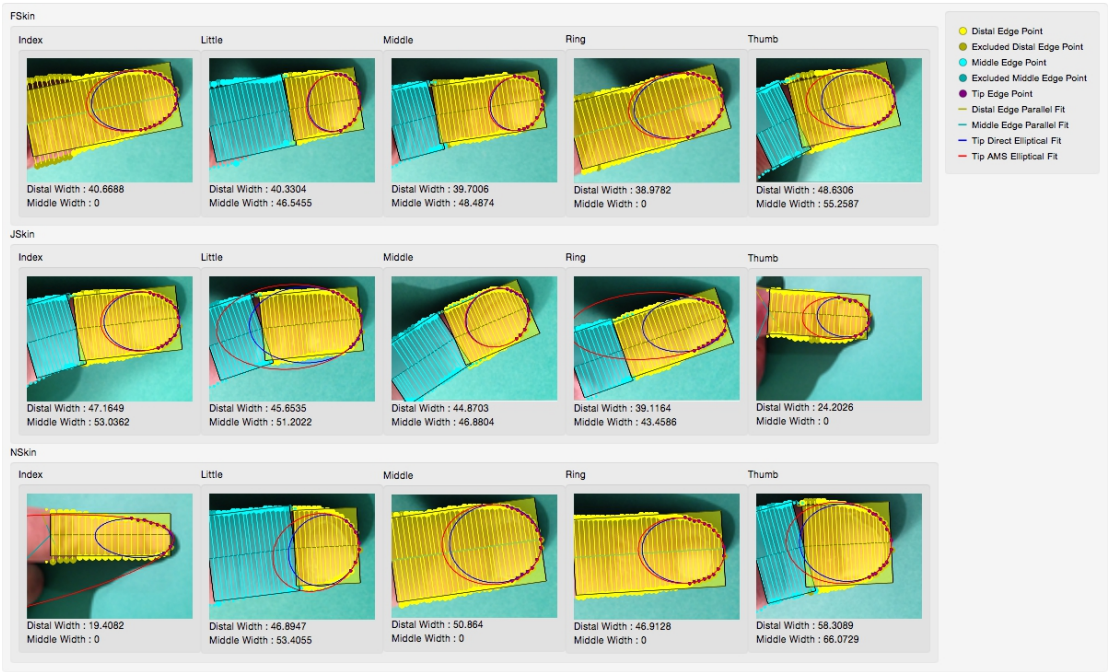


Fig. 4.3 Initial shape-fitting stage, with the Kink Fit, midline, and Parallel Edge fit, with AMS and Direct elliptical fits to the tip.

4.2.1.8 Set the Scale Space

We wish to set the mid-scale such that the number of pixels across the distal segment is close to a chosen target value. As mentioned in the scale space subsection 4.2.1.1 we are restricting the choice of scales to integer factors of the original and integer multiples of the small scale. For the iPhone images the factors are:

$$scale = 2^{-i}3^{-j}17^{-k} \qquad i \in \{0,1,2,3,4\} \qquad j \in \{0,1\} \qquad k \in \{0,1\} \tag{4.1}$$

The small scale space is made using the largest number of small factors as possible to achieve the desired image size. This gives the greatest flexibility in scaling the mid-size.

				0	1	2	3	4	i
k	k_{scale}	j	j_{scale}	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	i_{scale}
0	1	0	1	2448	1224	612	306	153	w
				3264	1632	816	408	204	h
		1	$\frac{1}{3}$	816	408	204	102	51	w
				1088	544	272	136	68	h
1	$\frac{1}{17}$	0	1	144	72	36	18	9	w
				192	96	48	24	12	h
		1	$\frac{1}{3}$	48	24	12	6	3	w
				64	32	16	8	4	h

Table 4.1 The image pixel dimensions for all possible integer factor scalings.

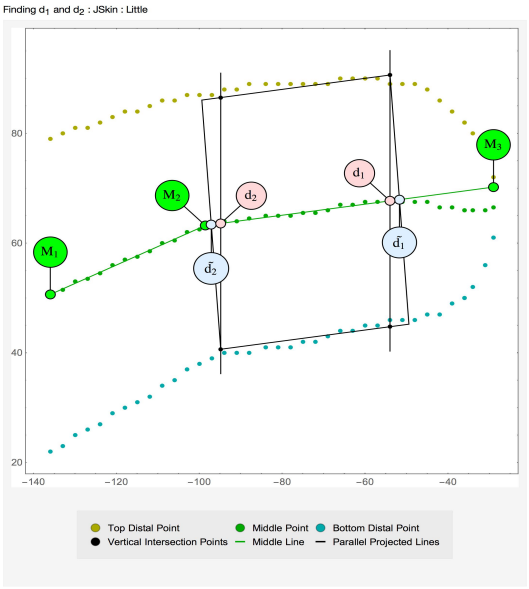
So for a chosen small scale $S_s = 2^{-3}3^{-1}$, a distal width in the small scale w and a minimum distal width in the medium scale w_m. Finding i_m and j_m which minimise $i_m + \frac{\log(3)}{\log(2)} j_m$ and which satisfy $2^{-i_m}3^{-j_m} - \lceil \frac{w_m}{w} \rceil \geq 0$, $i_m \leq i_s$ and $j_m \leq j_s$ determines the medium scale space.

4.2.1.9 Modelling the Fingertip

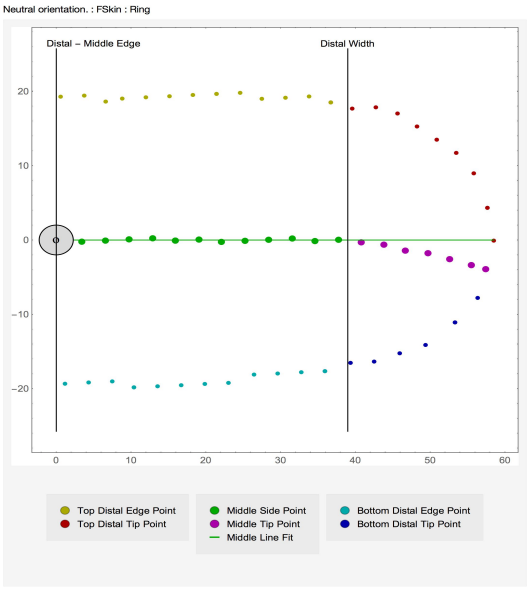
Having determined a measure of the width of the distal portion of the fingertip allows for a more robust model of the fingertip to be determined. This is because the measurement of the distal width is far more reliable than the distal length which has so far been determined, as the distal length has been found by evaluating where the "kink" of the midline points lies. For example, if the distal joint is unflexed, the knuckle portion will be overlooked entirely, as seen in Figure 4.3. It turns out that the anatomical ratios (Section 4.2) are a far more reliable method of determining the distal length.

From the kink fit, we have a midline, which is specified by three points. We now determine two points on this line, which are $\frac{1}{2}$ the distal width (d_1) and $1\frac{1}{2}$ the distal width from the tip (d_2). So the midline point that lies between the two points can now be taken to correspond to the straight-edged section of the distal portion of the finger. These points are passed to the Trapezian Fit method (E.0.5), and the points which lie between the tip and the point half the distal width from the tip are passed to the Elliptical Fit (E.0.4). The ellipse is adjusted to correspond with the two near vertices of the Trapezian; the result can be seen in Figure 4.4.

To illustrate the model, we can overlay the model on its source image in the original orientation and position. This shows that the application of the model actually only requires a position and orientation, because we assume that the camera is positioned such that the

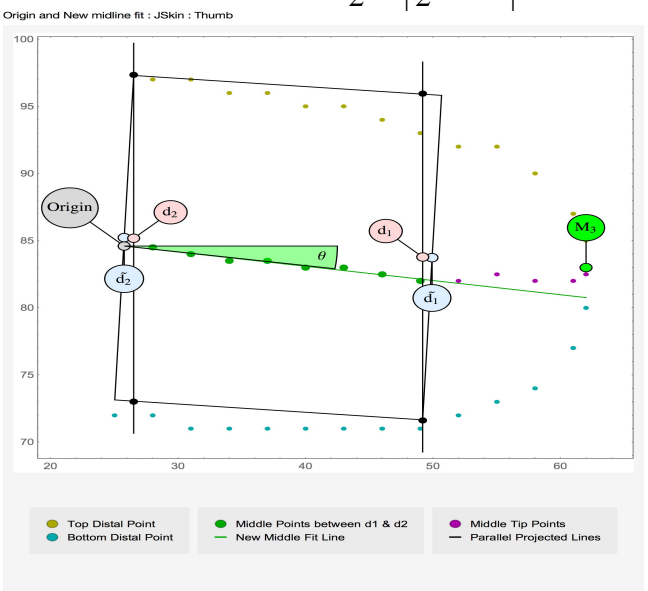


b) A New Midline Fit — We take the set of midline points, extract the subset which lies between d_1 and d_2, and then apply a straight line fit to these points. The origin is the fit evaluated at the x component of d_2, and we determine an orientation θ, which is arctan the gradient of the linear model.

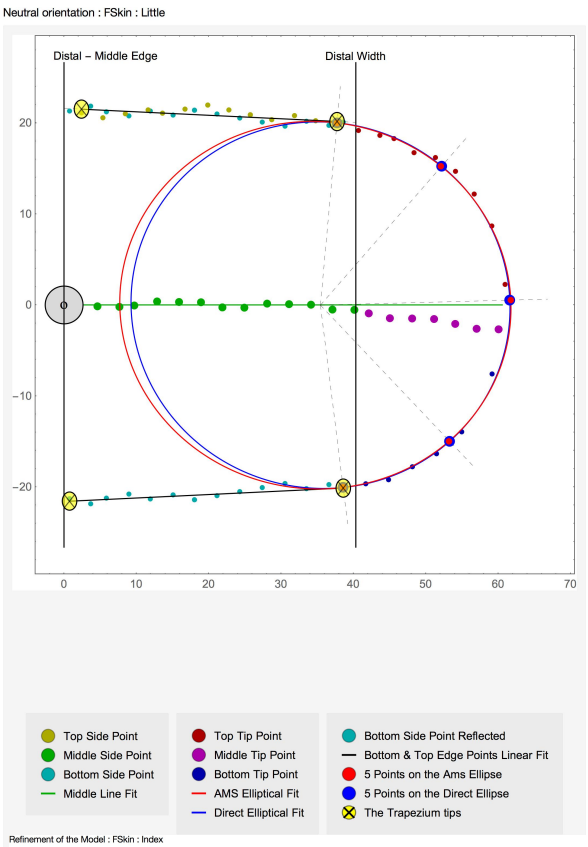


a) Determining d_1 & d_2 — We have a midline which is given as two points M_2 & M_3; one somewhere on the digit (M_2), and the other at the tip (M_3). If the finger were straight in the frame, then $d_1 = \tilde{d}_1$ and $d_2 = \tilde{d}_2$ could easily be found by finding the points on that line. However, the fact that the digit can be presented at an angle means that the edge points vertically above and below the point on the midline may include part of the curved tip. This has the effect of making these midpoints diverge from the midline of the digit. This can be corrected by adjusting with half the gradient of the midline $\delta\, mid$ to the ratios.

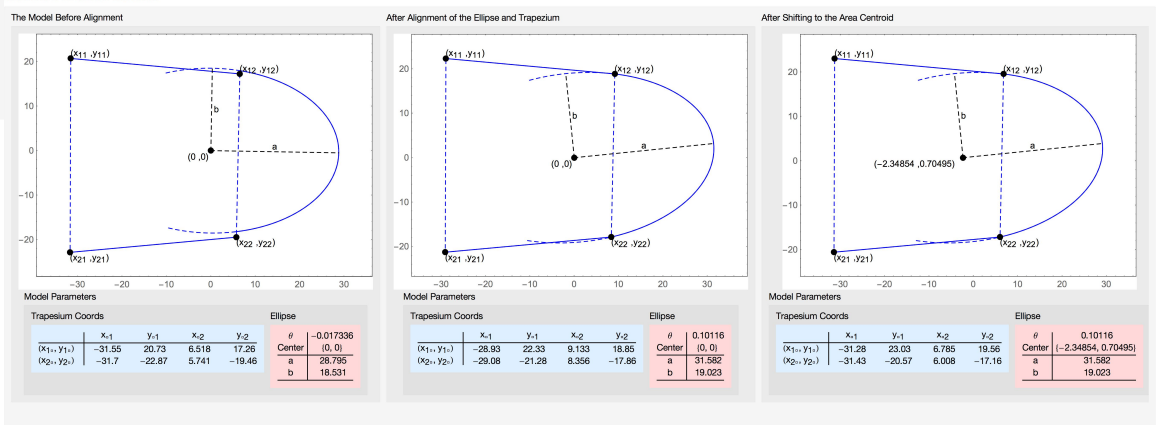
$$distance\, on\, line = (\frac{1}{2} + \left|\frac{1}{2}\delta\, mid\right|)w$$



c) Determine the Neutral Coordinates — We move all the points — the top and bottom edges, and the midline points — and we translate to the new origin. Then, we rotate by $-\theta$. This we call the "neutral coordinates", as the tip is aligned with its midaxis lying on the x axis. These points are separated at one distal width from the tip; a Trapezian fit is applied to one set of points, and an Elliptical fit is applied to the other. As a refinement to this, although the linear fit is applied as in the Trapezian section, the algorithm then looks at the points beyond and before the one distal width region; if they're consistent with the fit, then they are considered still part of the straight edge. The Elliptical fit is applied as discussed previously.



d) Fitting the Shapes — The comparison between the elliptical fit methods is done by looking at the distance from the front two points on the trapezium to the edge of the ellipse, and the best of these is chosen. Unfortunately, this is often not sufficient to make the ellipse join with the trapezium exactly because further on in the algorithm, these shapes will be used to perform a mask; we want to avoid introducing artefacts which could be interpreted as features. So, we wish to remove these discontinuous intersections between the ellipse and the trapezium. In order to do this, we choose three points on the Elliptical fit and the two front points of the trapezium, thereby making a set of five points, and we pass these through the direct Elliptical fit algorithm. It should be noted that five points is what is required to completely define an ellipse; there is only one ellipse which will pass through five points.



e) Finalizing the Model — The coordinates for the ellipse and the trapezian are adjusted so that the origin is at the center of the ellipse, and the ellipse has zero rotation. The reason for this is that it makes calculating the distance from the edges of the ellipse easier. So, now when we apply the model to the image, all that's required is a position — which is the position of the center of the ellipse — and an orientation about that position.

Fig. 4.4 The Fingertip Model Algorithm.

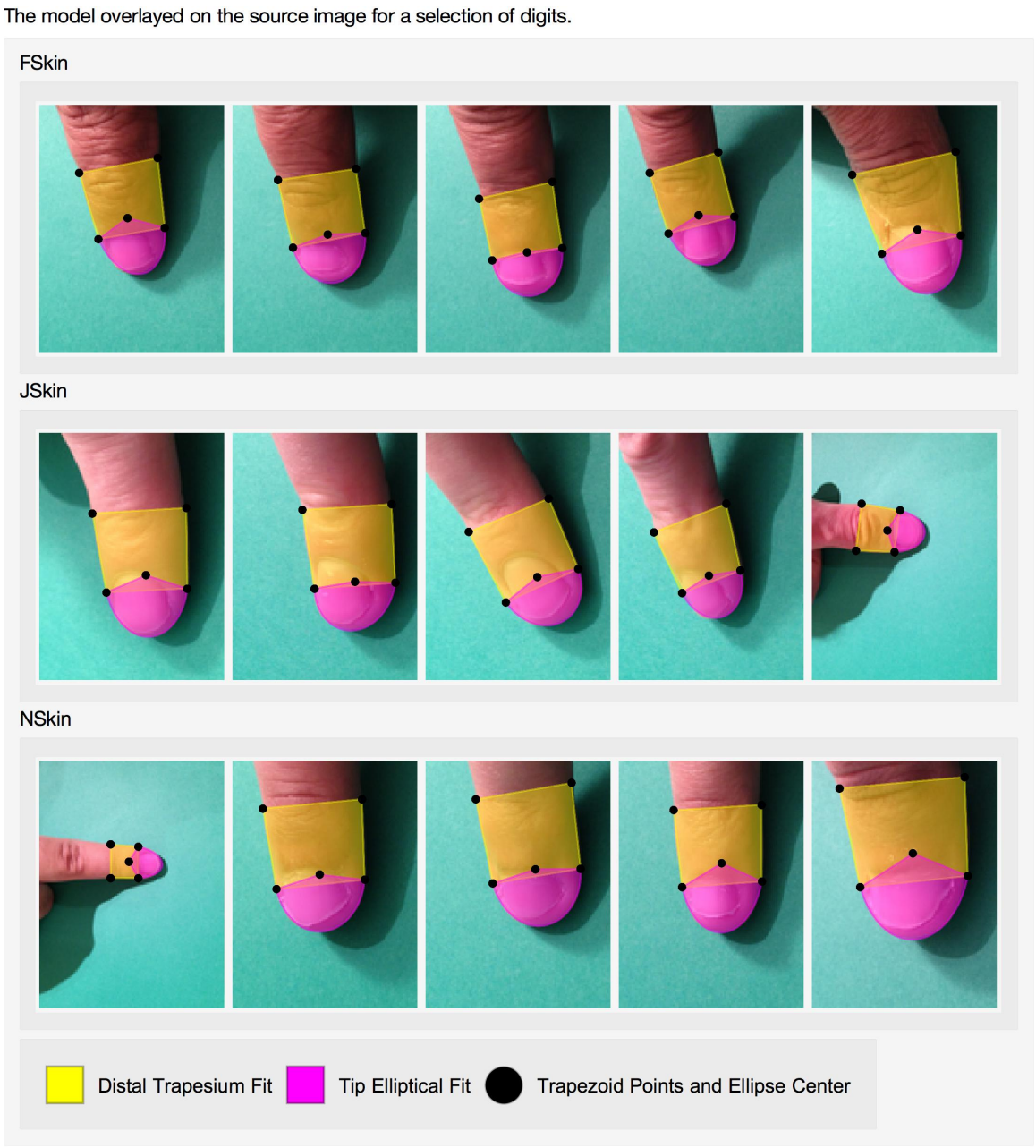


Fig. 4.5 The models overlayed on the source images.

digit presents at the same scale when in contact with a surface throughout the frame; we're not accounting for perspective effects in this simple application. The model can be seen overlayed on the selection of digits for three individuals in Figure 4.5.

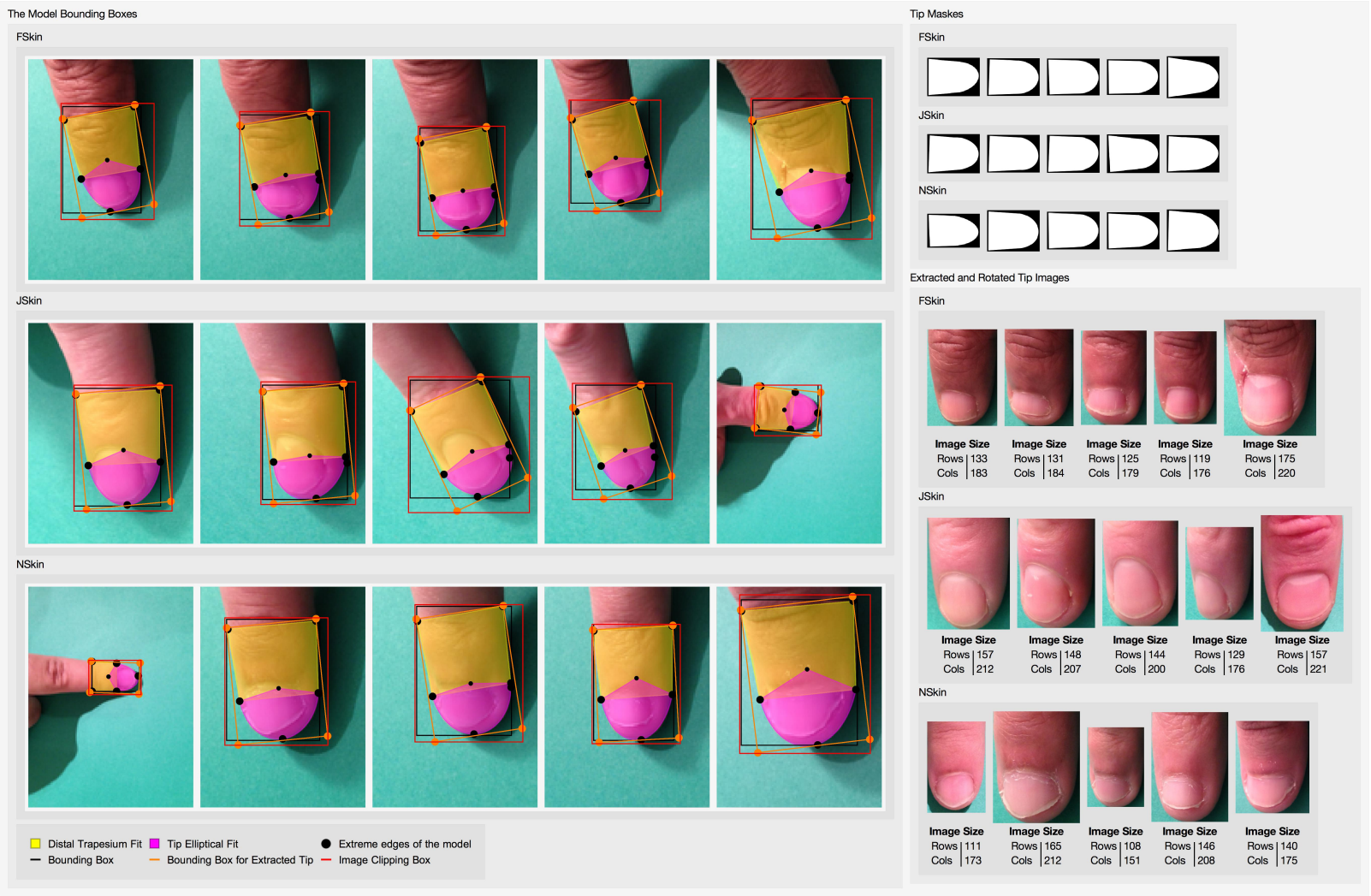


Fig. 4.6 The models and model boundaries overlayed on the source images. With tip masks and extracted tip images.

4.3 Dynamic Tracking

The main purpose of the dynamic tracking algorithm is to categorize the motion of the digit in the frame as "rapid motion," "smooth motion," and "in contact with a surface."

- **Rapid motion** — When the digit is being moved into the frame, or is moving rapidly across the frame, or when the camera is being put into position, it is pointless to attempt to detect a fingertip as this information will quickly become irrelevant. Standard motion detection library routines can be used here on the unaltered video feed. When these algorithms detect motion above a certain threshold, the algorithm does nothing and waits for the motion detected to fall below this threshold.

- **Smooth motion** — Smooth motion is when the digit is being moved in a controlled manner within the frame. When the digit is moving in such a way, it is reasonable to assume that a finger press is imminent. The algorithm takes a frame in the small scale space and processes it into a pixel-categorized image as in Section D.2.1. The frame orientation and digit tip selection are found as in Sections 4.2.1.2 and 4.2.1.3, and then a Filament fill of only the tip is performed similar to Sections E.0.2 and 4.2.1.4, but now with the Filament fill path being shortened to the length of the distal section in the model. The position and orientation for the model are obtained using the Force Analogue Shape Detection algorithm outlined below, in Section E.0.6. The algorithm continues tracking until the change in the position of the tip between successive frames falls below a threshold, at which point the algorithm decides that the digit is likely in contact with a surface, and control passes to the "in contact with a surface" part of the algorithm.

- **In contact with a surface (ICWaS)** — When the tip of the digit is relatively immobile, the digit is likely in contact with a surface; we now want to observe the blood flow. The frame is now captured in the medium space, and is cropped to the edges of the model in its current position and orientation. The cropped image is put into the skin color space, and a gradient filter is applied to the grayscale channel. If this is the first iteration, this gradient-filtered image is stored; if it is a subsequent iteration, this gradient filter is compared with the first and an alignment transform is found between the current frame and the first frame. To improve the accuracy and avoid the algorithm being distracted by movements at the edge of the digit, the gradient-filtered images are masked with the model, which removes edge-related gradient features, focusing

the algorithm on the nail, which is stable during a finger press. The transform is applied to the two chromatic channels, which aligns the images, and the difference is found between successive frames; this gives us the blood flow.

4.3.1 Rapid Motion Tracking

Before any fingertip tracking is performed, the algorithm checks to see if the digit is still in motion. This is done by first taking the difference between subsequent grayscaled frames from the raw video feed using OpenCV's 'absdiff' method. To account for changes in lighting and contrast, a binary threshold is applied to the frames, after which an erosion operation (the 'erode' method) is performed to further reduce artifacts and false positives. If the number of changes in the image is above a given threshold, then the digit is considered to be in rapid motion.

4.3.2 Smooth Motion Tracking

The find-tip algorithm (Section 4.2.1.3) gives us a path which we know lies on a digit. This path, however, is not necessarily particularly well-aligned with the axis of the digit. For this reason, we don't ask for a shortened path from the tip to $1\frac{1}{2}$ distal widths along that path; such a path might actually end closer to the tip than desired, as the path isn't necessarily along the axis. To account for this, we ask for two distal widths from the tip.

The next problem to be addressed results from the operation of the Filament fill method; because the digit isn't necessarily presented perpendicularly to the frame edge and the Filament fill algorithm finds points perpendicular to the frame edge, we adjust by half the gradient of the path, similarly to the method used in the fingertip model algorithm illustrated in Figure 4.4. So, Filament fill is run along the find-tip path from the tip to a distance of $(2+\left|\frac{1}{2}\delta path\right|)w$ from the tip. This gives us three sets of points: the top, middle and bottom edges, which definitely include points relating to the tip of the digit.

Now we refine these points; because we know the orientation of the frame, we can determine which points correspond to the top edge of the digit, and which points correspond to the bottom. We now put them into standard orientation as in steps a and b in Figure 4.4.

We now need to find points on the model which would ideally correspond with the points found by Filament fill in the image. First, we need to obtain a reasonably good measure of the orientation of the digit; given that we have midline points on the digit which reasonably correspond to the distal segment, a linear fit to the middle $\frac{2}{3}$ of these points provides a good measure of the orientation with little computational cost. Rather than reorientating

the image points to a neutral coordinates, we rotate the model to correspond with the image coordinates. This is easily achieved as the model is centered on the ellipse, and so involves a rotation of the four points of the trapezian and adding the rotation to the ellipse's orientation.

In the model's new orientation, we find the furthest point on the model in the x axis. (It should be noted that this is not necessarily what we would refer to as the "tip" of the digit.) We assume that this point on the model corresponds to the first point in the midline set of points; the Filament fill follows the path from the tip back to the frame edge, so the first point in the midline should be the tip. So, we create a set of x axis coordinates where the first midline point corresponds with the furthest point on the model in the x axis. We do this because the Filament fill divides up the shape vertically in a relatively regular fashion, so we are dividing up the model in the way that we would expect Filament fill to if the model were an image. This allows us to put points on the model and points in the image into a $1:1$ correspondence. So although the fingertip model is a relatively arbitrary geometric shape, the regularity of the Filament fill method allows us to use a force analogue shape detection algorithm as outlined previously (Section E.0.6).

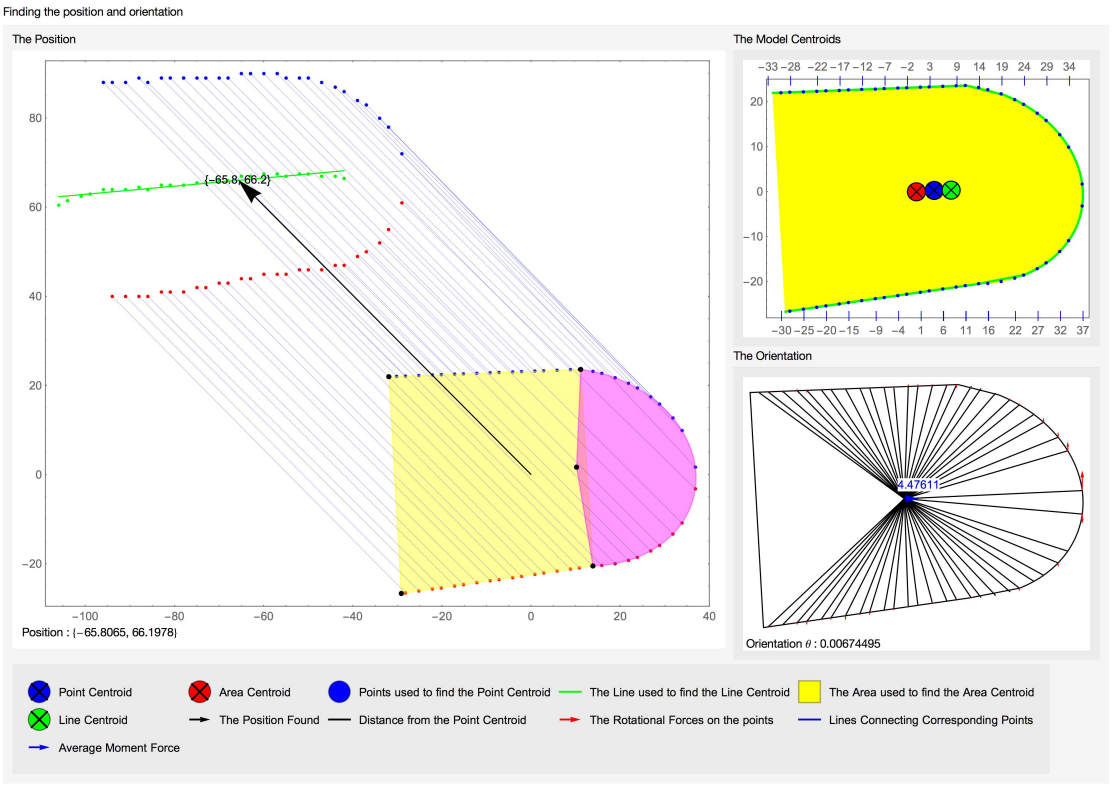


Fig. 4.7 Finding the Position and Orientation of the Tip. A side-effect of the pre-alignment of the model means that, almost always, the change in the orientation from the force analogue shape detection algorithm is negligible.

A side-effect of the pre-alignment of the model means that — almost always — the change in the orientation from the force analogue shape detection algorithm is negligible.

The distribution of points around the perimeter of the model is dependant on the orientation of the digit in the image. This means that the point centroid is not constant, and so a shape-based area centroid is used in the modeling stage (Figure 4.7).

In practice, the algorithm first finds the rough orientation of the digit in the image and rotates the model accordingly. The model is re-centered on the point centroid for the current orientation before the FASDM method is used.

In addition to the FASDM method, the algorithm also finds the mean of the length of the vectors between the rotated and translated model points and the points on the image. To be clear, this distance is signed according to which point is further away from the model center. This allows the model to be scaled. The upshot of this is that the code correctly follows a digit even as the digit moves closer or further away from the camera. (Fingers do not normally change shape.)

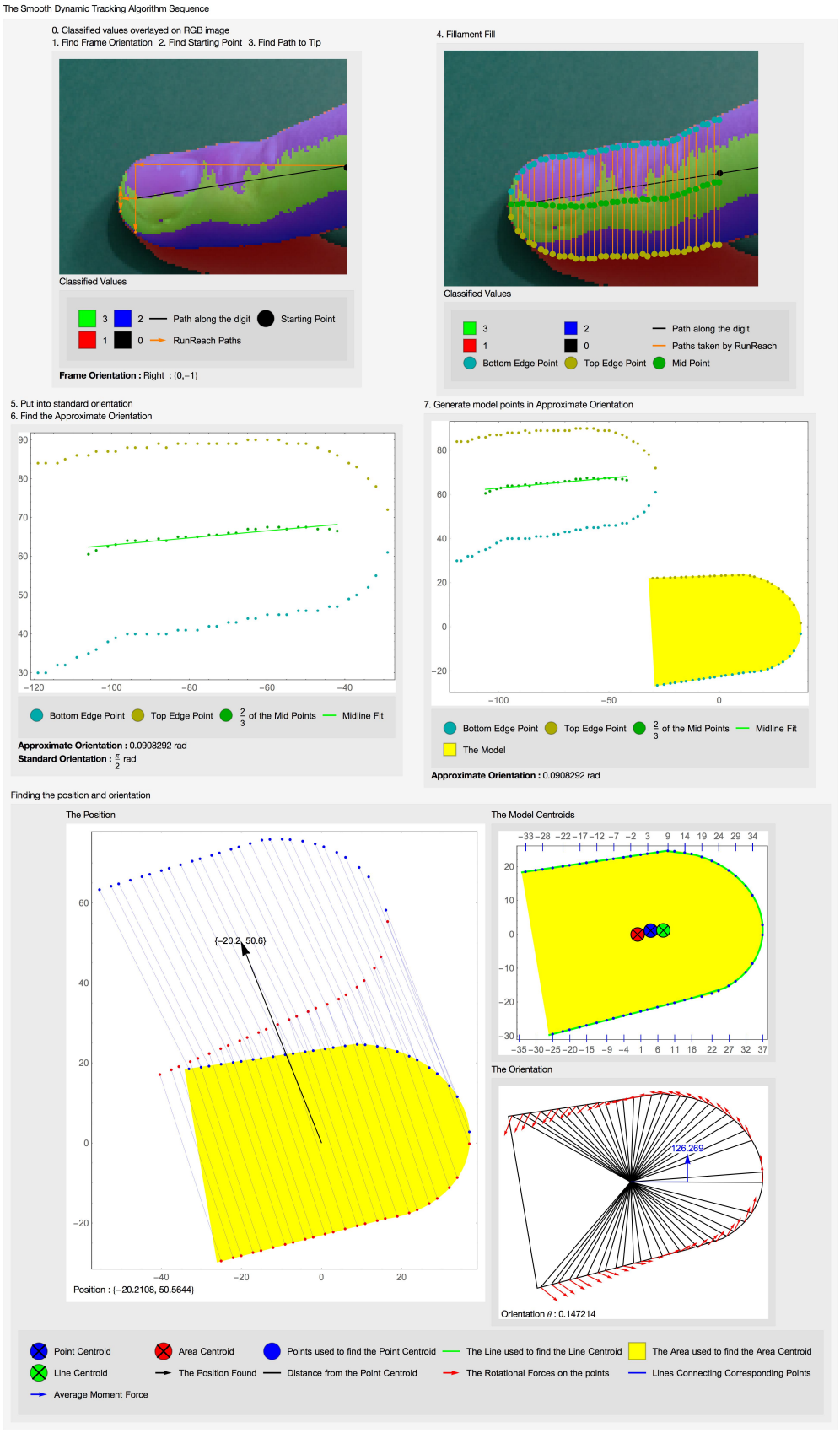


Fig. 4.8 Smooth Dynamic Sequence

4.3.3 ICWaS Method

The ICWaS method performs three main tasks: cropping the midscale image around the tip; providing a multiplicative mask which highlights blood flow in the tip; and aligning successive frames in-order to find the pixel color differences in the skin color space. Combining these three tasks allows the blood flow to be shown.

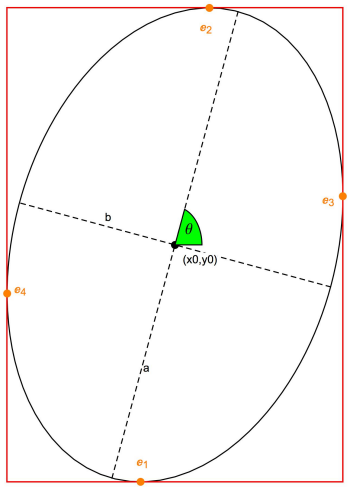
4.3.3.1 ICWaS Setup

From the Smooth Movement Tracking (Section 4.3.2), we have a model position and orientation in the small scale space. We wish to track the blood flow using the medium scale image, so first we must put the model in the medium scale space. This is as simple as multiplying the coordinates by a scaling factor as shown in Section 4.2.1.1.

Now we want to find the extreme top, bottom, left and right edges of the model in order to crop the midscale image appropriately. The model is comprised of a quadrilateral and an ellipse. To find the extreme edges of the quadrilateral, the algorithm just finds the minimum and maximum values of the coordinates of its vertices. The ellipse's extreme edges can be found using

$$
\begin{aligned}
e_1 &= \left(x_0 + \tfrac{(a-b)(a+b)}{A}\sin(\theta) \quad , y_0 + A\cos(\theta) \right) \\
e_2 &= \left(x_0 - \tfrac{(a-b)(a+b)}{A}\sin(\theta) \quad , y_0 - A\cos(\theta) \right) \\
e_3 &= \left(x_0 - B\cos(\theta) \quad , y_0 - \tfrac{(a-b)(a+b)}{B}\sin(\theta) \right) \\
e_4 &= \left(x_0 + B\cos(\theta) \quad , y_0 + \tfrac{(a-b)(a+b)}{B}\sin(\theta) \right)
\end{aligned}
$$

$$
\text{where} \quad A = \sqrt{a^2\tan^2(\theta) + b^2}, \quad B = \sqrt{a^2 + b^2\tan^2(\theta)}, \quad -\tfrac{\Pi}{2} < \theta < \tfrac{\Pi}{2}
$$



However, we only want to include extreme points which lie on the curve which models the tip. The curve which models the tip is the part of the ellipse which falls between the two points on the quadrilateral closest to the tip; we can check if the extreme edges of the ellipse are on this part of the curve by checking that they lie at an angular position between the angles at which the tipmost quadrilateral points lie.

Now we have the extreme edges of the model. These points are used to find a bounding

box, which includes all the tip pixels in the small scale space. Translating these to the medium scale space, we subtract the scaling factor from the lower coordinates and add the scaling factor to the higher coordinates, ensuring that the box does not crop any pixels from the digit in the medium scale space. We now take the medium scale RGB image, crop it to the model bounding box, and then pass the extracted tip image to the skin color space processing routines. The grayscale image is saved as a template for the template matching algorithm.

During a finger press, we expect the flesh to deform around the tip, making the edges of the digit unreliable for alignment, so we wish to construct a binary mask which tells the template matching algorithm which pixels to use for alignment. Using OpenCV's drawing routines, we render the model in the midscale tip image coordinates. This is done in a way which actually renders the model 'slightly' by four or five pixels smaller than the model proper. This effectively eliminates the edge of the digit next to the background, pushing the algorithm to rely on features which do not deform, such as the nail.

4.3.3.2 Blood Flow

We assume that the digit's movement will not take it outside of the bounding box set up in the initialization step (Section 4.3.3.1), so we capture the tip image from the video feed. The RGB tip image is converted to the skin color space, then the grayscale channel image is used to align with the previous captured frame; if this is the first pass through the loop, this is the image captured in the initialization. The frame alignment metric is masked using the binary rastered model image generated in the initialization.

The image alignment pixel shift updates the change in the model position since the first frame. If the change in the model position since the first frame is greater than the ICWaS threshold, then the control passes back to the Smooth Motion routine. The current frame is aligned with the previous frame and the difference is found between the pixel values in the chromatic channels; this is the blood flow (Figures 4.9, 4.10 & 4.11). These steps are repeated until the change in the model position since the first frame causes the routine to transfer control back to the Smooth Motion routine.

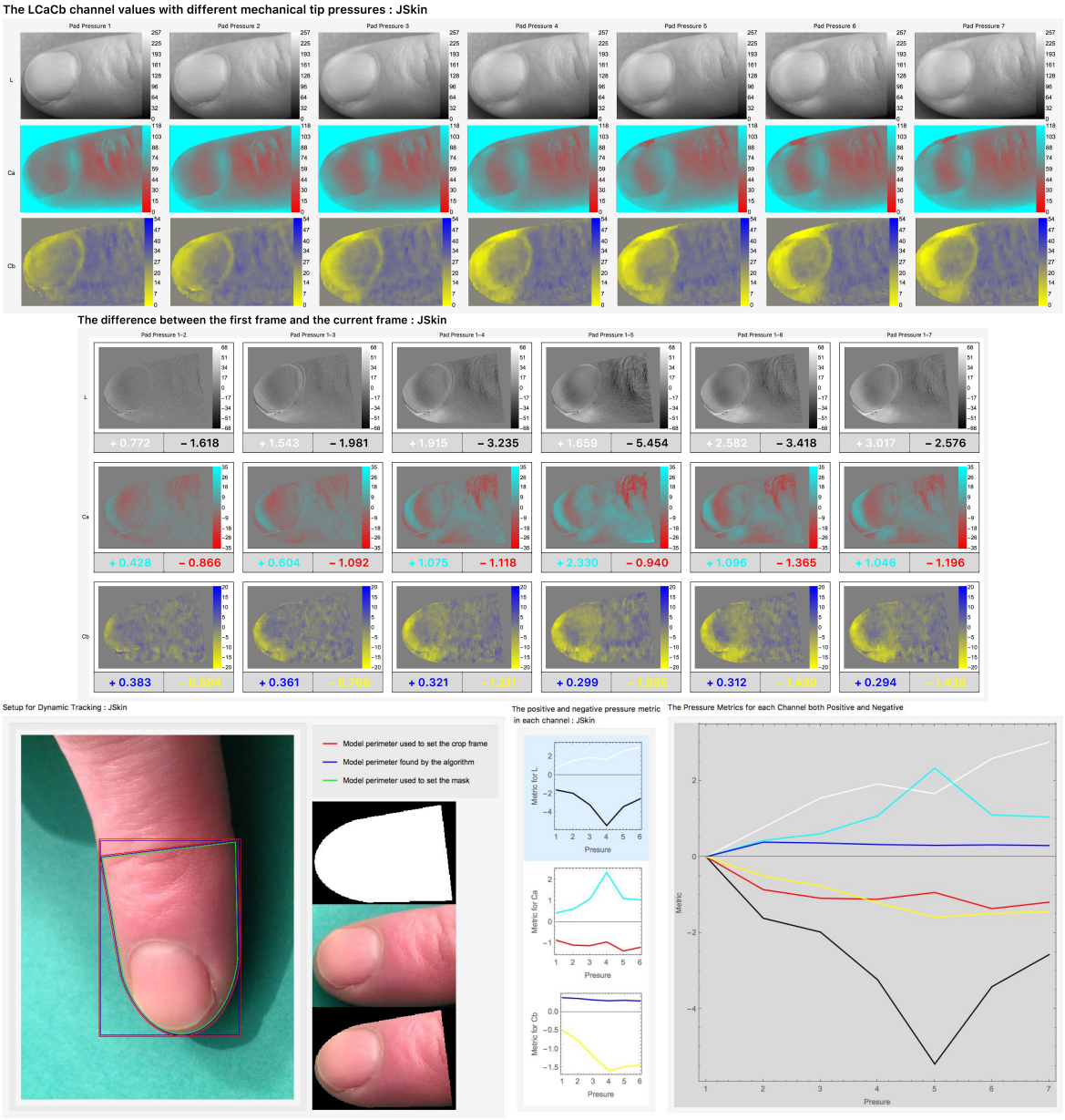


Fig. 4.9 The resulting sequence of images from the ICWaS algorithm followed by the differences. The metric is the result of summing the positive and negative changes separately and then dividing by the number of pixels. Finally, the initial image of the sequence is shown with the model outline, the slightly larger model used to set the crop frame and the slightly smaller model used for setting the mask.

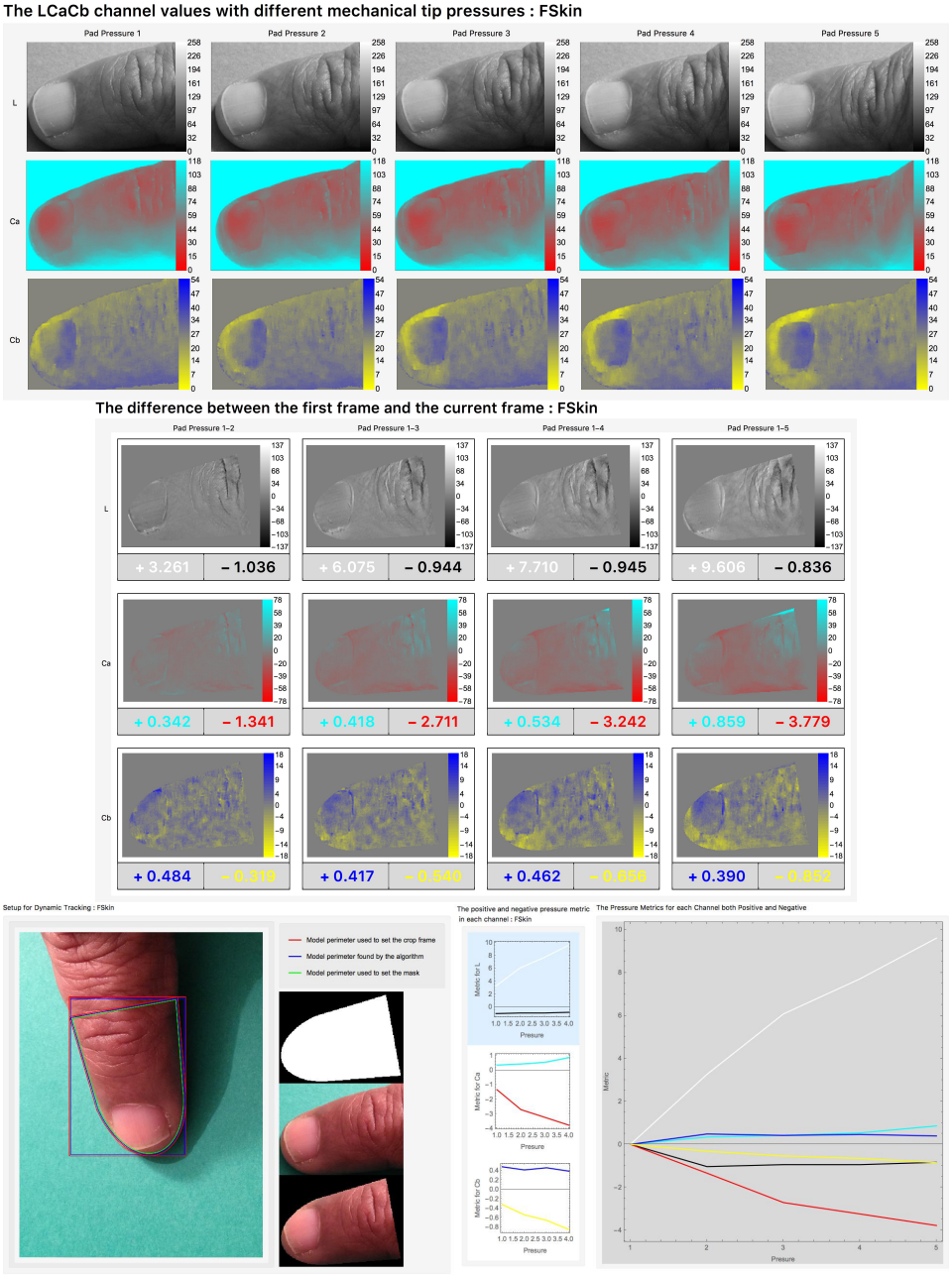


Fig. 4.10 The resulting sequence of images from the ICWaS algorithm followed by the differences. The metric is the result of summing the positive and negative changes separately and then dividing by the number of pixels. Finally, the initial image of the sequence is shown with the model outline, the slightly larger model used to set the crop frame and the slightly smaller model used for setting the mask.

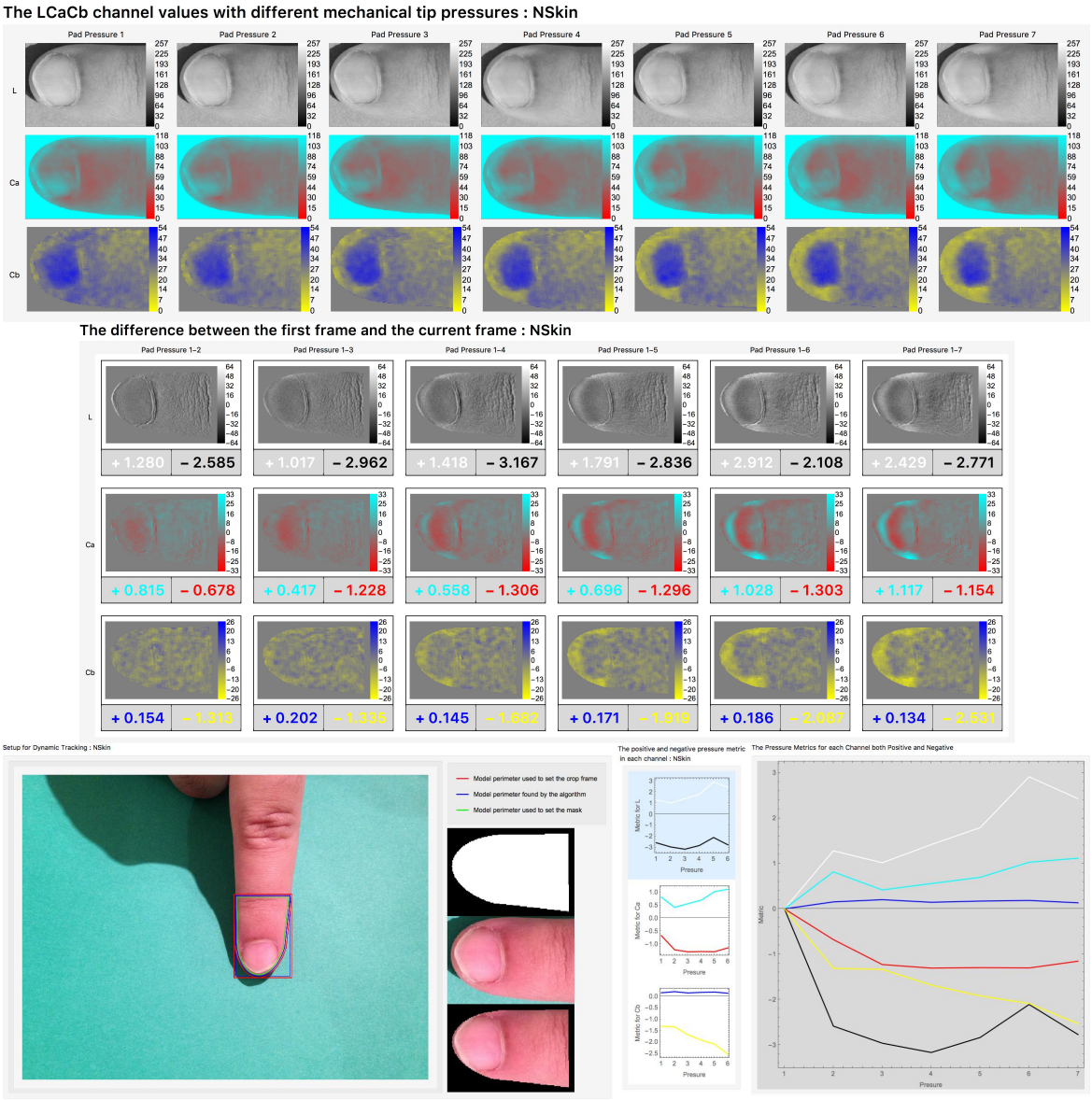


Fig. 4.11 The resulting sequence of images from the ICWaS algorithm followed by the differences. The metric is the result of summing the positive and negative changes separately and then dividing by the number of pixels. Finally, the initial image of the sequence is shown with the model outline, the slightly larger model used to set the crop frame and the slightly smaller model used for setting the mask.

4.3.3.3 Blood Flow Metric

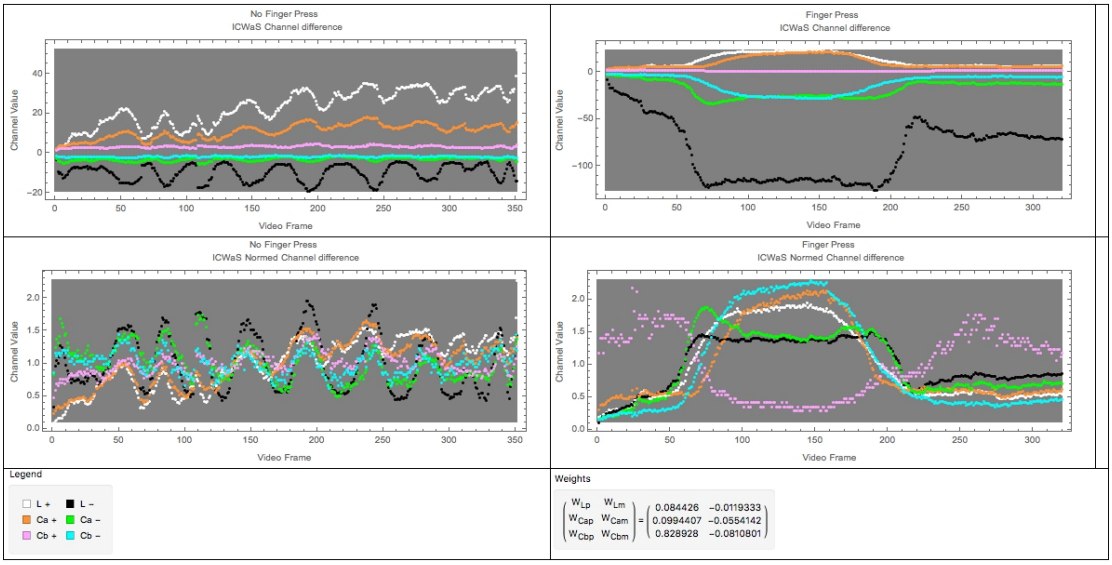


Fig. 4.12 The normalized 6-channel metrics, revealing the shape of the distributions and facilitating numerical techniques.

We have a 6-channel metric from the ICWaS method which we wish to use to detect the presence of mechanical stress in the tip, i.e. the finger tip pressing on the surface. The simplest solution is to form a linear combination of the 6-channel metric and to set a threshold above which the finger tip is considered to be pressing on the surface.

The metric needs to be able to distinguish between the finger sliding slowly across the surface and the finger pressing on the surface. Two ICWaS runs are compared in Figure 4.12, one with the finger sliding slowly across the surface ('No-Press' data), and the other with the finger pressing on the surface then releasing pressure ('Press' data).

First, the 6-channel metrics are normalized to reveal the shape of the distributions and to facilitate numerical techniques (Figure 4.12). Then we fit a function, which is a combination of a straight line with a Gaussian, to the two sets of data, as seen in Figure 4.13.

$$f^{set}_{channel}(x) = Ae^{-\frac{0.5(x-\mu)^2}{\sigma^2}} + mx + c \tag{4.2}$$

The 6 Press data functions clearly exhibit a common peak and a similar variance, as illustrated in Figure 4.14. This allows the Press data to be divided into regions where we want the combined metric to register minimally and maximally. It is also desirable for the combined metric to be flat when no pressure is being applied. These considerations allow a set of equations to be defined:

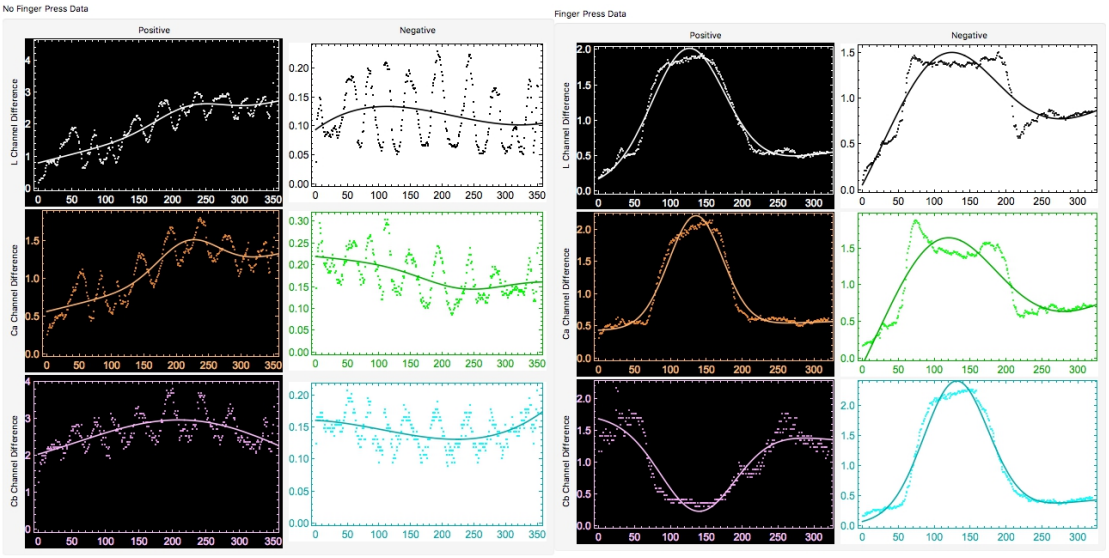


Fig. 4.13 A functional fit to the 6-channel metrics for both the 'Press' and 'No-Press' data. The fit uses a combination of a Gaussian and a straight line.

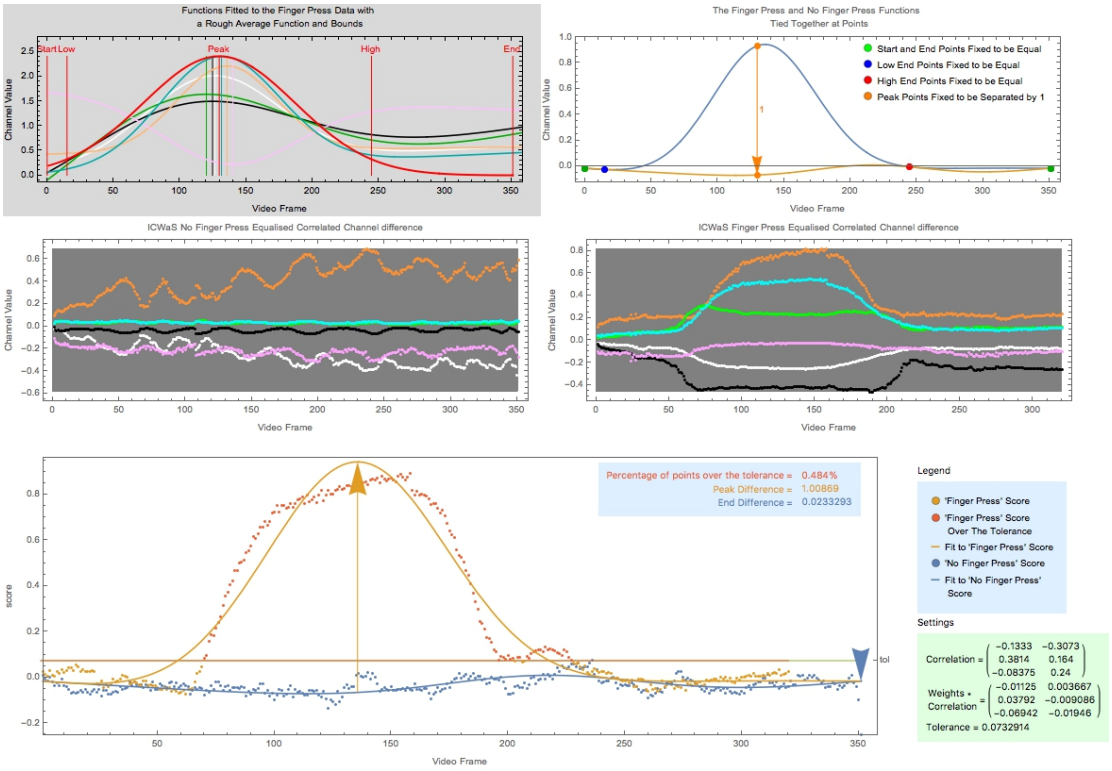


Fig. 4.14 The Analytic Correlation. Fixing the points for the linear combination, as seen in top right diagram, an analytic solution is found for the correlation matrix.

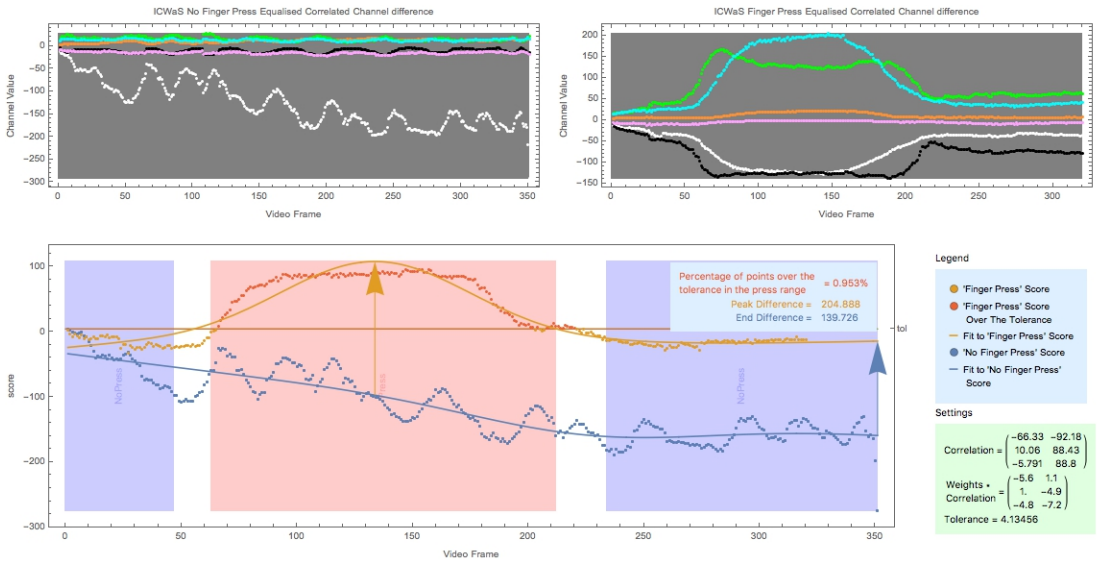


Fig. 4.15 The numerically-found correlation matrix.

$$
\begin{aligned}
F^{np}(x) = C_{Ca^-}F^{np}_{Ca^-}(x) + C_{Ca^+}F^{np}_{Ca^+}(x) + \\
C_{Cb^-}F^{np}_{Cb^-}(x) + C_{Cb^+}F^{np}_{Cb^+}(x) + \\
C_{L^-}F^{np}_{L^-}(x) + C_{L^+}F^{np}_{L^+}(x) \\
F^{p}(x) = C_{Ca^-}F^{p}_{Ca^-}(x) + C_{Ca^+}F^{p}_{Ca^+}(x) + \\
C_{Cb^-}F^{p}_{Cb^-}(x) + C_{Cb^+}F^{p}_{Cb^+}(x) + \\
C_{L^-}F^{p}_{L^-}(x) + C_{L^+}F^{p}_{L^+}(x) \qquad (4.3)
\end{aligned}
$$

$$
\begin{aligned}
F^{np}(x_{start}) = F^{p}(x_{start}) = F^{np}(x_{end}) = F^{p}(x_{end}) \\
F^{np}(x_{low}) = F^{p}(x_{low}) \\
F^{np}(x_{high}) = F^{p}(x_{high}) \qquad (4.4)
\end{aligned}
$$

These conditions (Equation 4.4) can be solved for finding the 6 linear coefficients (Equation 4.3.3.3). The results can be seen in Figure 4.14.

For completeness, a numerical technique to find the 6 linear coefficients was also attempted. The success was scored by checking the 'correct' classification of the data as Press or No-Press. The results can be seen in Figure 4.15.

The two different methods were run using data from different individuals and different digits.

4.3.4 Dynamic State Transitions

For each of the three states, we need to define a metric which characterizes the motion detected. Using these metrics, we need to define four tolerances for the state transitions.

The metric for the rapid motion detection is the average pixel value of the eroded difference image (Section 4.3), so if two consecutive images are the same, then the metric is 0. And if every pixel has changed by its maximum amount (e.g. every pixel black/white), then the metric is 255. The only question remaining is what sort of values should we expect for rapid finger movement. We can roughly estimate that a finger occupies about 10% of the frame; given that the background is illuminated by the same amount, we can expect pixel differences around 20%. Thus, although the range of the metric is actually 0-255, we actually expect the tolerance to be quite low.

The metric for the smooth motion detection is the pixel distance moved by the model divided by the distal width, as seen in Section 4.3.2. Significant values for this metric are 0 if the model hasn't had to move; 1 if the model is moved by a finger width; and it is assigned a negative value when the finger is not detected, moved out of frame, or if an insufficient amount of the digit remains in the frame. So, for negative values, the model isn't updated.

The ICWaS detection method returns the distance that the bounding box for the fingertip has had to move to align the image with the previous frame (Section 4.3.3). OpenCV's alignment routine requires the images to be well-aligned in order to function properly, so before attempting alignment, the code initially checks for large movements and does not perform the alignment if such movement is detected, avoiding long iterations of OpenCV's routine which would ultimately result in an error. However, in order to make this robust, the routine also catches any error from the alignment routine. Both of these result in an negative value being returned.

In order to set reasonable values for the tolerances, a version of the code was written which, regardless of the state, performed both the rapid and smooth tracking routines. Using timed intervals, the code was run with the user performing motions which should be classified as a 'rapid movement, 'smooth movement' and 'ICWaS movement'. The results can be seen in Figure 4.17.

From this figure, you can see the metrics are fairly noisy; we want to avoid triggering a state transition on the basis of a single video frame. This problem is effectively solved in the code simply by taking the average of the current metric with the previous value, thus smoothing out the noise and stabilizing the state transitions, as illustrated by Figure 4.18.

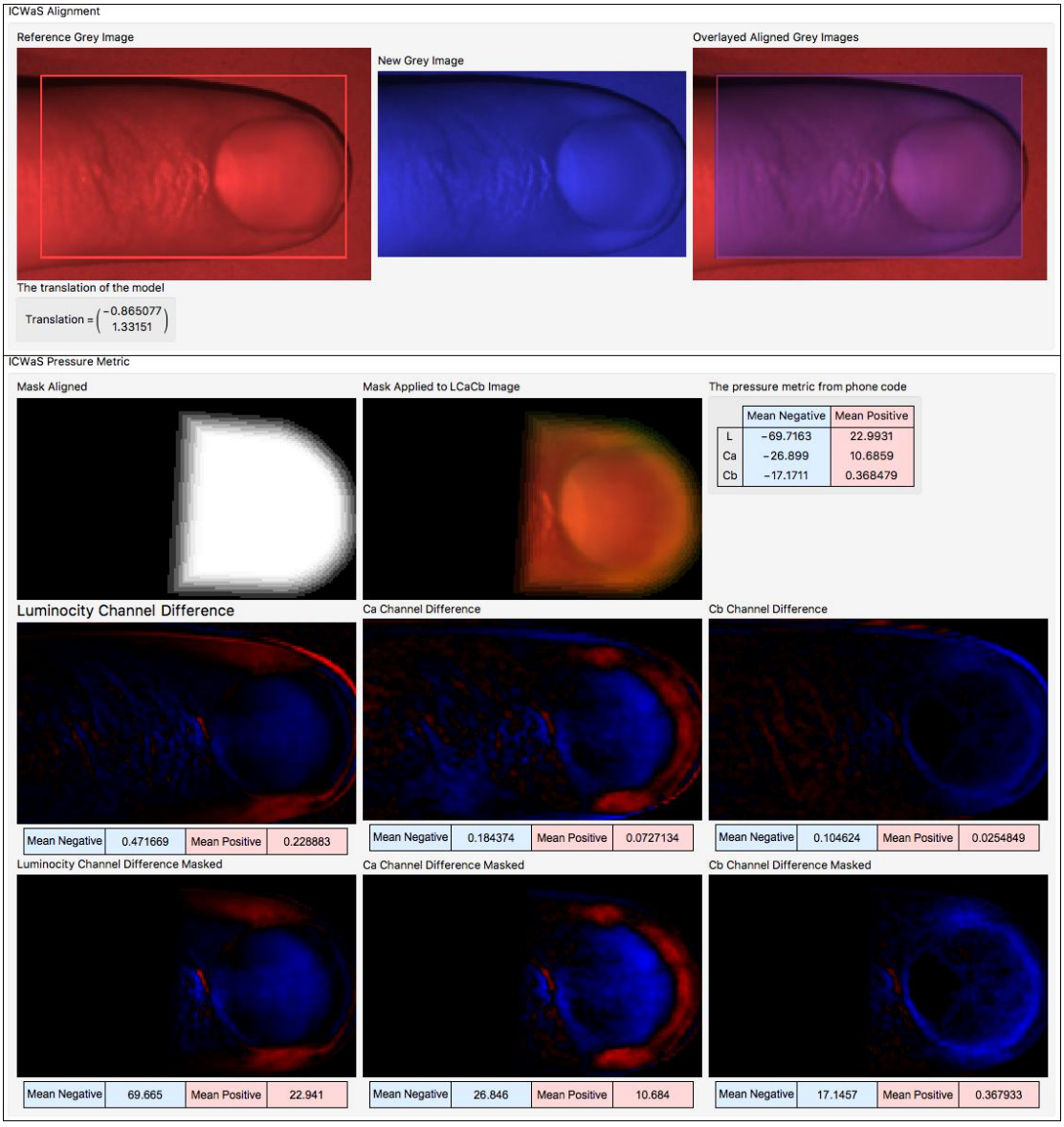


Fig. 4.16 ICWaS alignment and Metric

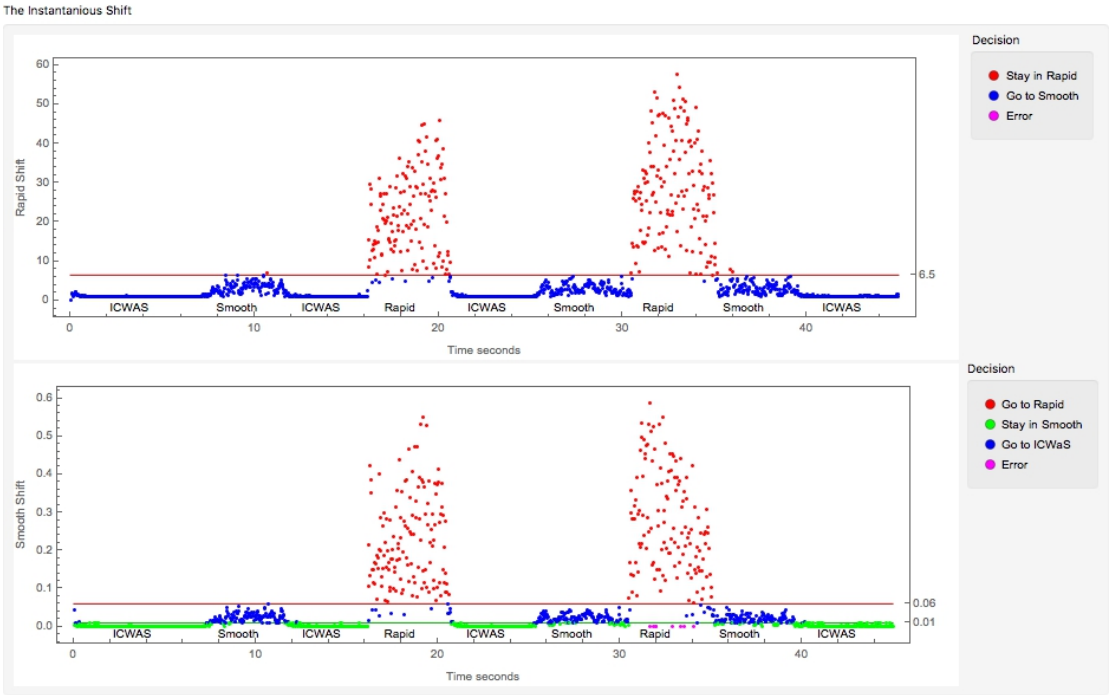


Fig. 4.17 The rapid and smooth dynamic tracking metrics.

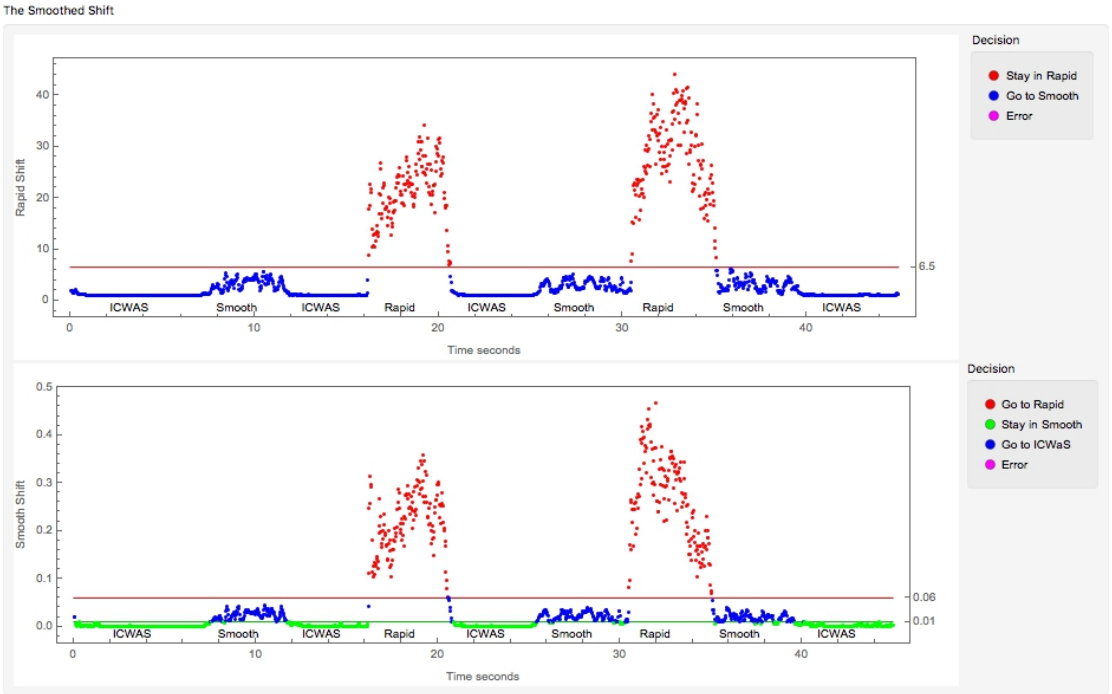


Fig. 4.18 The rapid and smooth dynamic tracking metrics with values smoothed by averaging with previous values.

4.4 iPhone Camera Setup

Quite later on in development, an iOS update was released which allows in-code control of camera properties, such as focus, exposure and white point color balance. Because the code does per-pixel comparisons and assumes that the color values remain constant, the first thing this allows us to do is fix the exposure, the white point and the focus such that, as the code is tracking the digit and tracking the blood flow, the algorithm isn't thrown off by the phone deciding to suddenly changing the camera properties.

Initially, a simple routine was added which simply fixed these camera properties before starting the tracking. However, it was apparent that we could take advantage of the camera control features to produce a better result. First, we needed to be able to perform a metric for how good the camera settings are for tracking. Initially, this metric was taken using the whole image. Taking the classified image, the metric was formed by taking a weighted sum of the classified image pixel values; the metric scored values of 0 and 3 positively and values of 1 and 2 negatively, the idea being to promote camera settings which result in a classified image which rewards certainty.

However, the problem with this approach is that if the camera properties are set with, say, a very low exposure, the image is almost entirely black and the classified image is certain that there isn't a finger in the picture, regardless of the reality, so this metric is scored highly. To combat this, the app allows two regions to be selected in the frame, one which definitely contains skin, and one which definitely does not. We then have two separate metrics for the skin region; values of 2 and 3 are scored highly, while values of 0 and 1 are scored negatively, and conversely for the skinless region.

After all this work, we have three metrics: one which rewards correctly identifying skin, one which rewards correctly identifying background, and one which rewards overall certainty. The final metric which was returned in the camera calibration is the weighted sum of these three metrics; the skin and the background identification metrics given a slightly higher weighting than the overall certainty.

4.4.1 Setting the Exposure

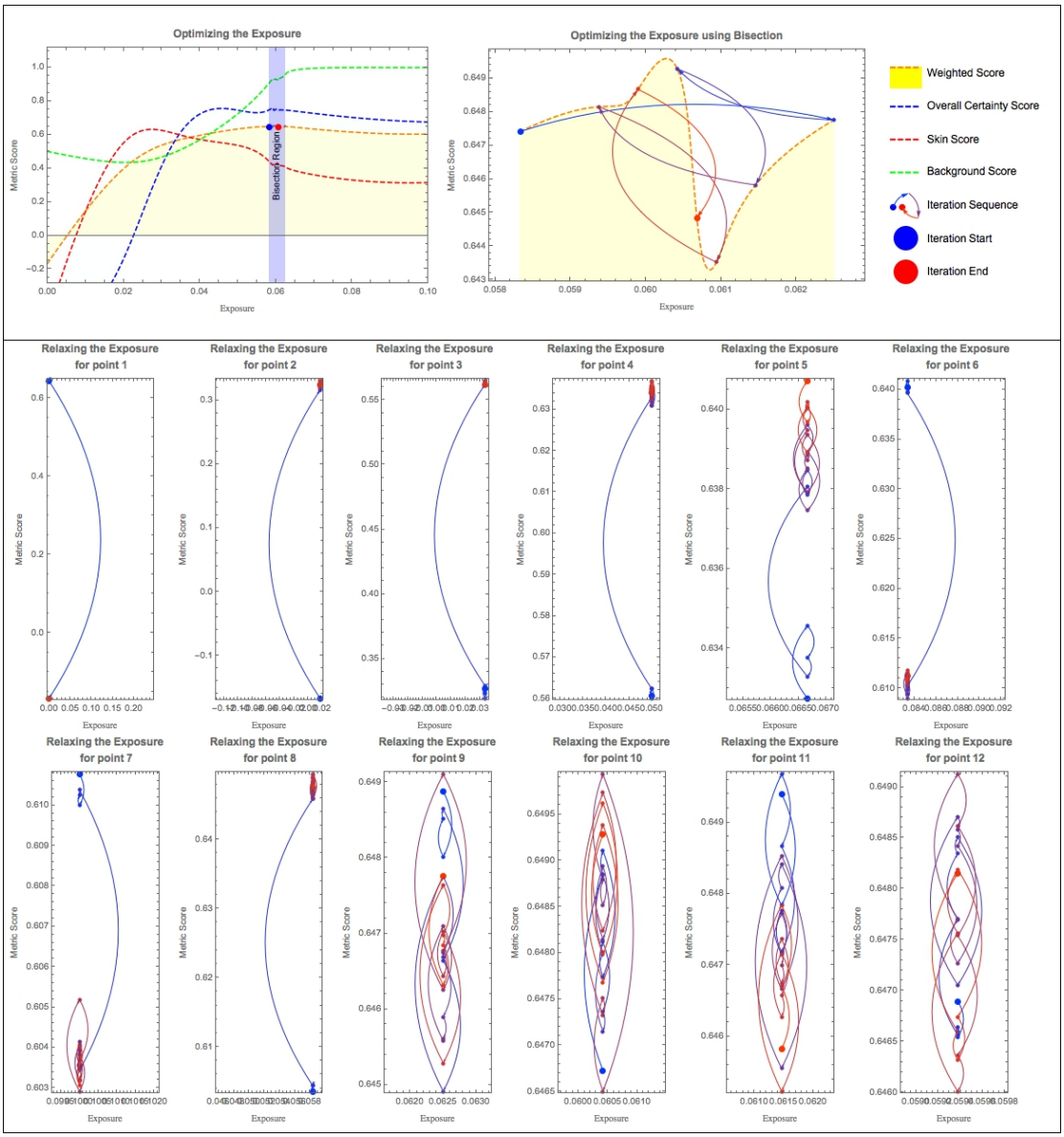


Fig. 4.19 Optimization of the exposure for the iPhone camera.

To find the optimal exposure, the code first picks a handful of values between a minimum and maximum exposure and evaluates the metric at these values. This is done by setting the exposure waiting in-code for a few microseconds for the camera to finish adjusting and then evaluating the metric. This is necessary because the camera properties are set in a different thread to the tracking algorithm's thread. A bisection algorithm is then used to refine the optimal exposure (Figure 4.19).

4.5 Putting it All Together

So far, we have said little about the app itself; the app associated with this project is designed to demonstrate how the fast color space transform developed in Chapter 2 can be used in conjunction with the simple shape detection routines developed in this chapter to create a simple fingertip mechanical-stress detector using an iPhone.

It is considered good practice when developing an iPhone app to split the code into three parts: the model, which handles all the computation; the view, which handles display and interprets user gestures; and the controller, which is the intermediary, essentially handling the control flow of the user interface. Each can be considered to be running in its own thread, and the Objective-C language handles the interaction between these threads. The recommended division of the control assigns each step in the program as indicated in Figure 4.20.

The chart accurately presents the logical flow and recommended assignment of the program elements. However, the chart fails to address how the model, controller and view are each running in their own threads, and so control never actually passes from the model to the controller, or vice versa. So, in the MVC structure, the model and the view may only message the controller to indicate that an event has happened. For instance, when processing a video stream, if every time the dynamic tracking part of the model required a new image, a request message must be sent to the controller, then it would need to wait for the controller to get the image from the video stream and send it to the model, i.e. the model would often be waiting for the controller to respond. For this reason, in practice, the model uses OpenCV's video stream handling capabilities to get an image directly from the video stream.

The charts in Figures 4.21, 4.22, 4.23 and 4.24 more accurately depict the actual implementation of the model by showing the data flow to the elements of the model which are accessible to the controller.

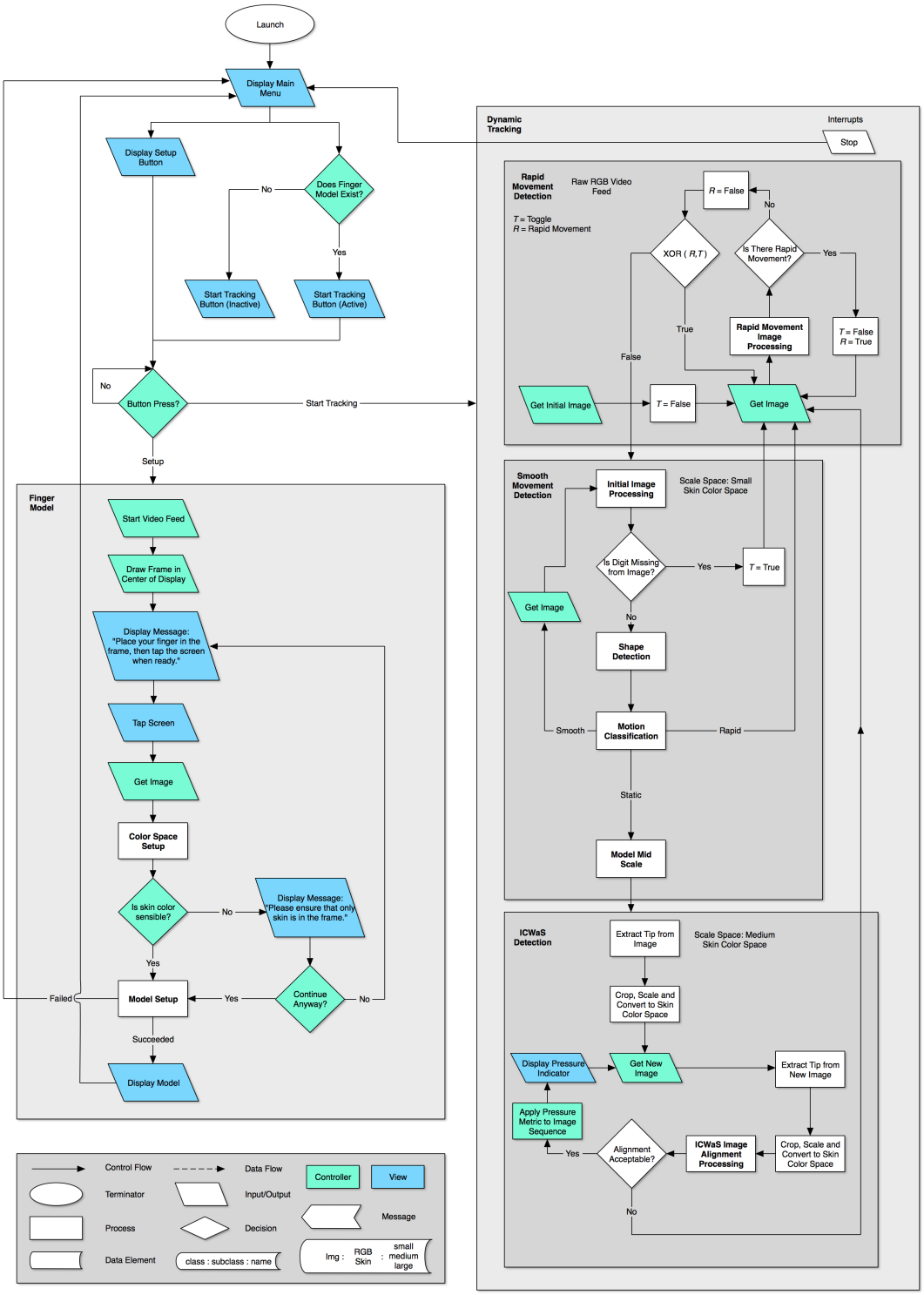


Fig. 4.20 Fingerpress UI flowchart

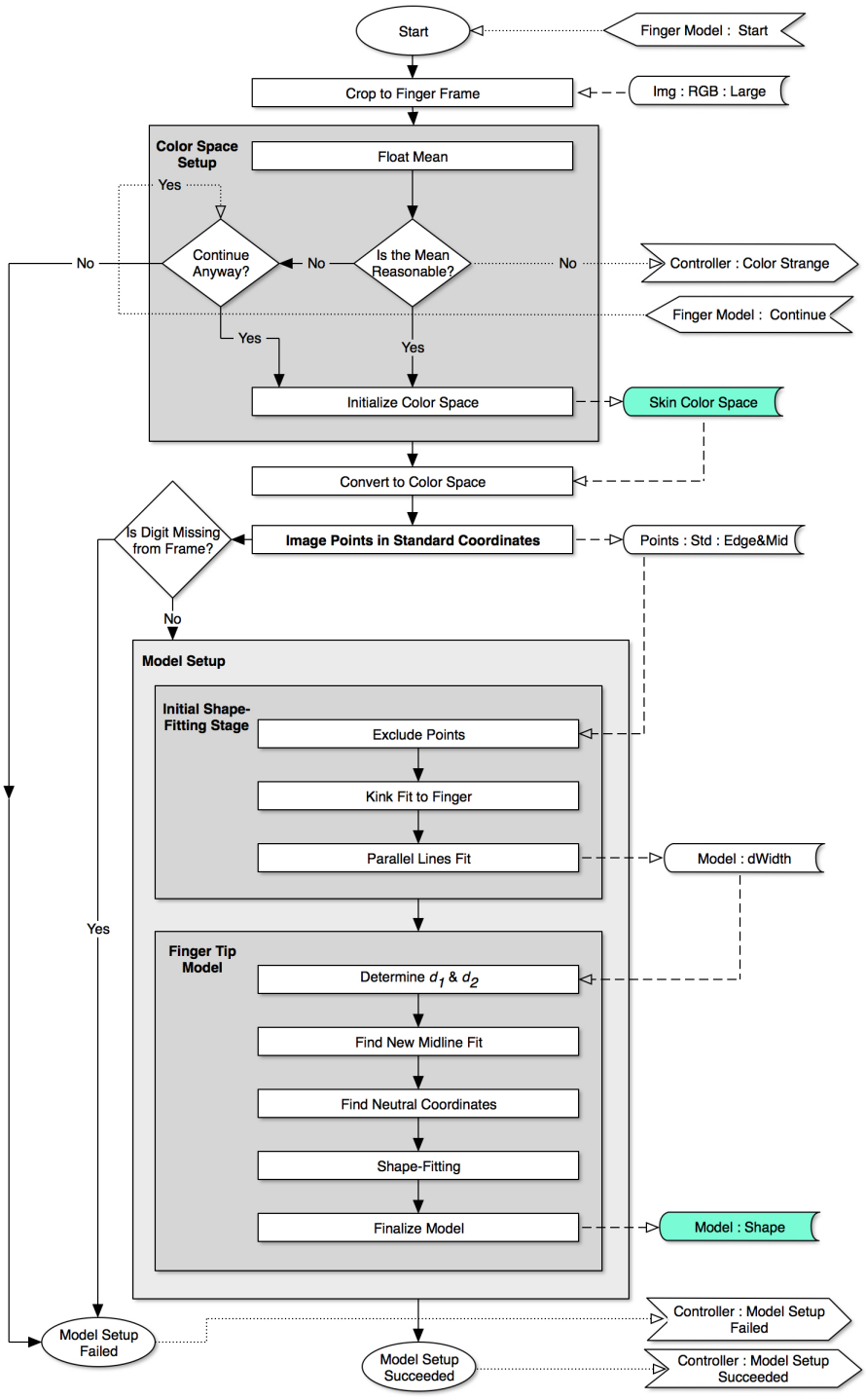


Fig. 4.21 Finger model flowchart

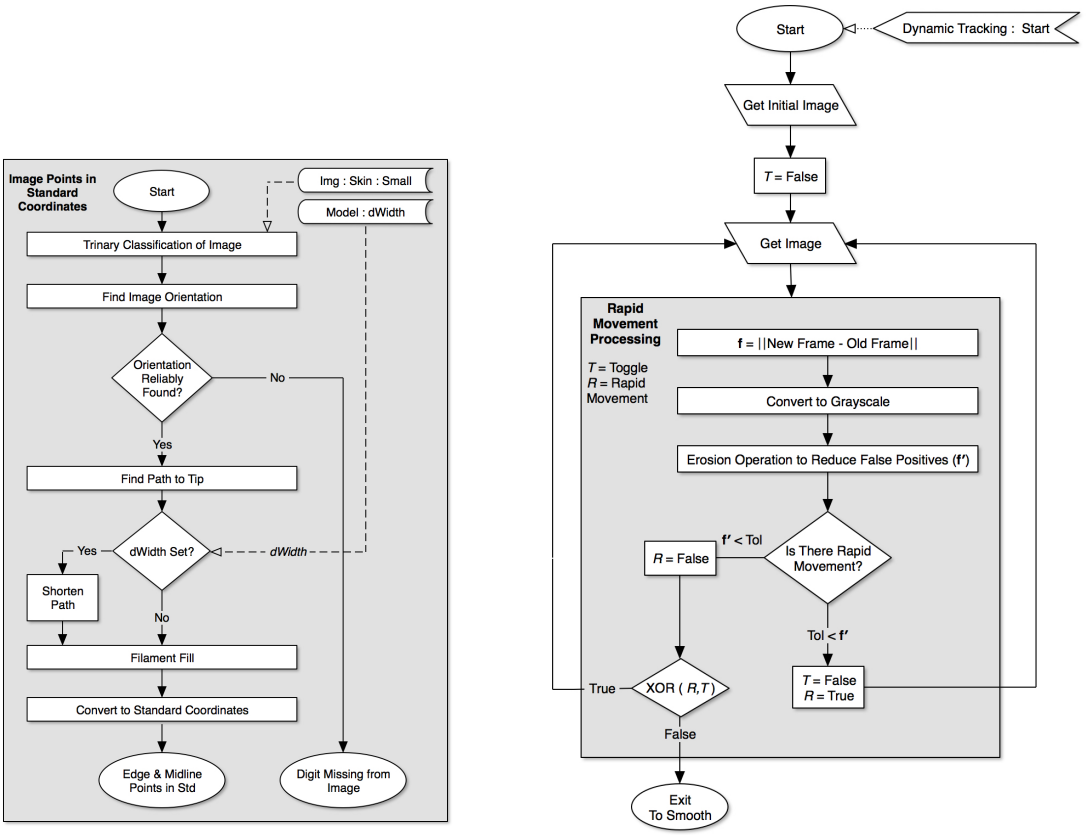


Fig. 4.22 Flowcharts for finding image points in standard coordinates (left), and rapid motion detection (right).

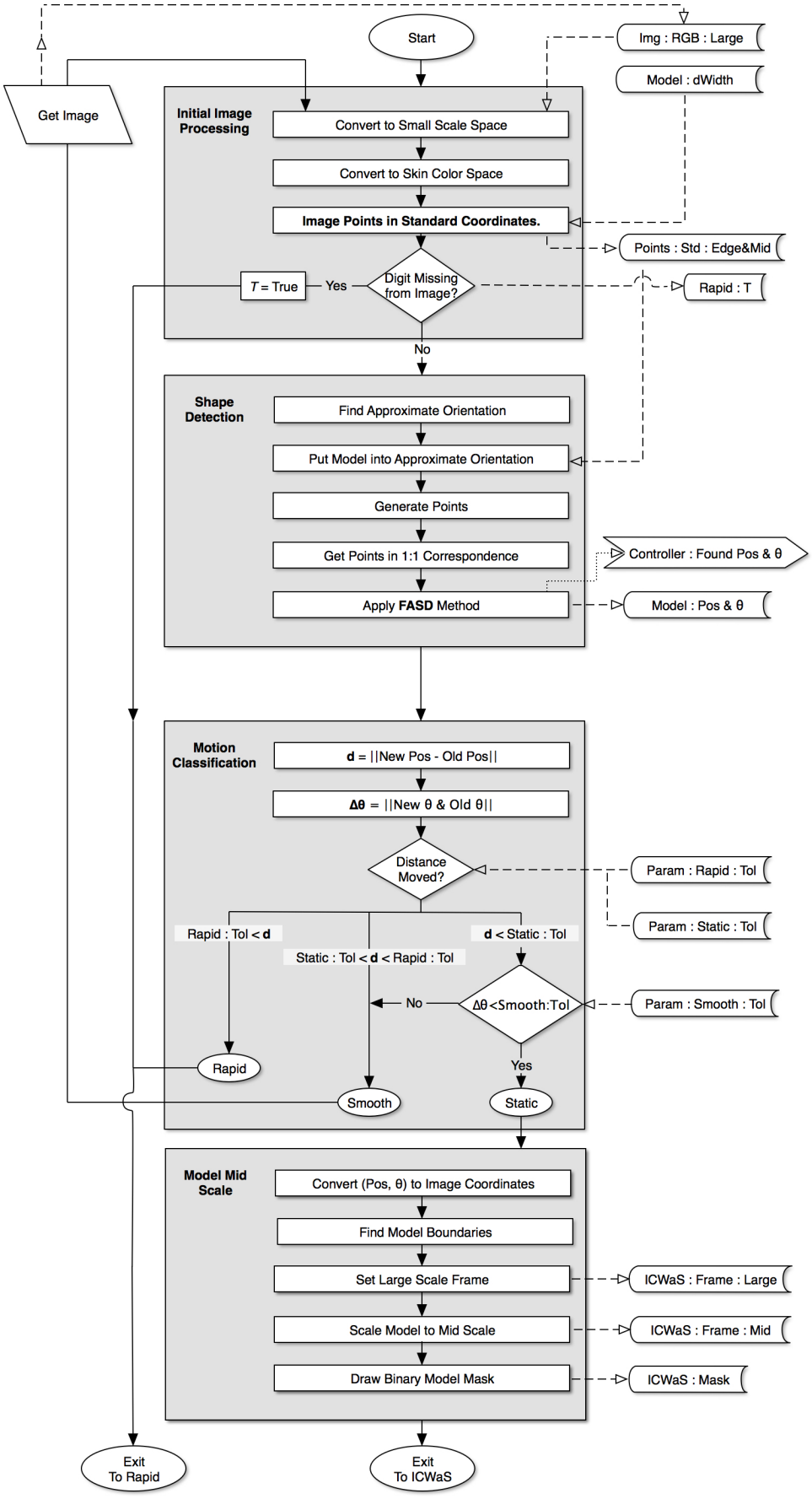


Fig. 4.23 Smooth movement flowchart

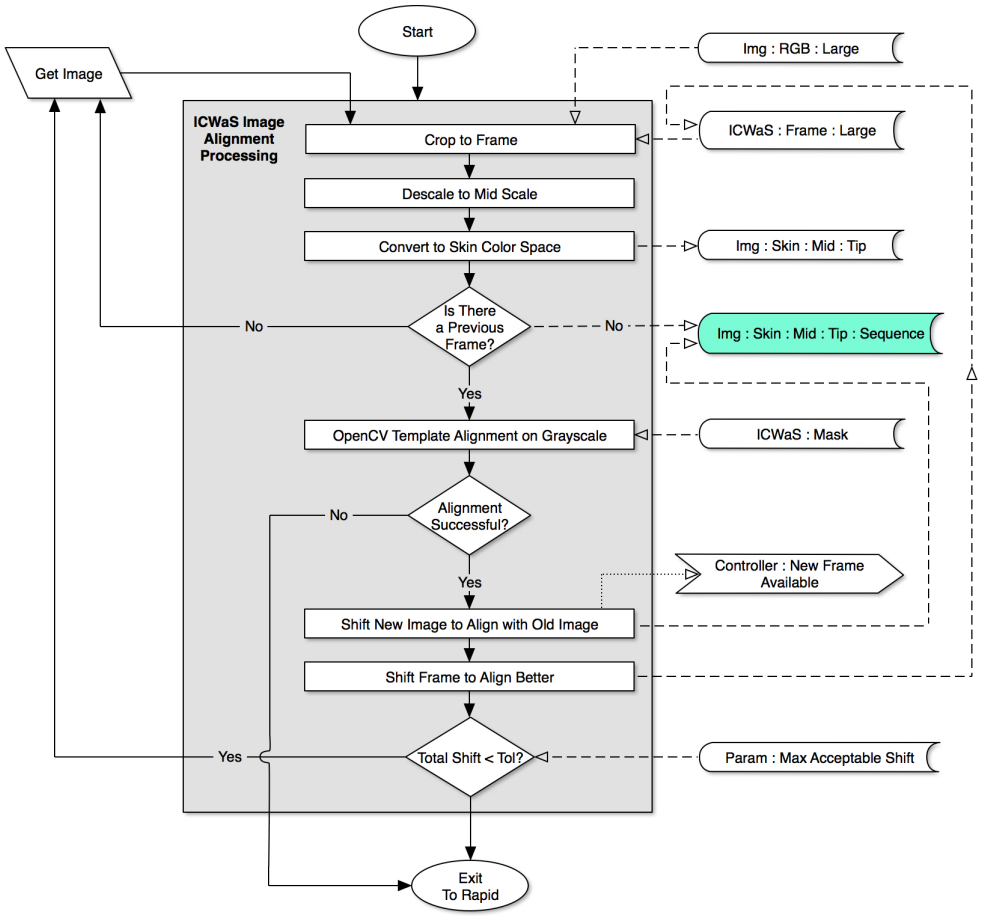


Fig. 4.24 ICWaS flowchart

4.6 Appendix

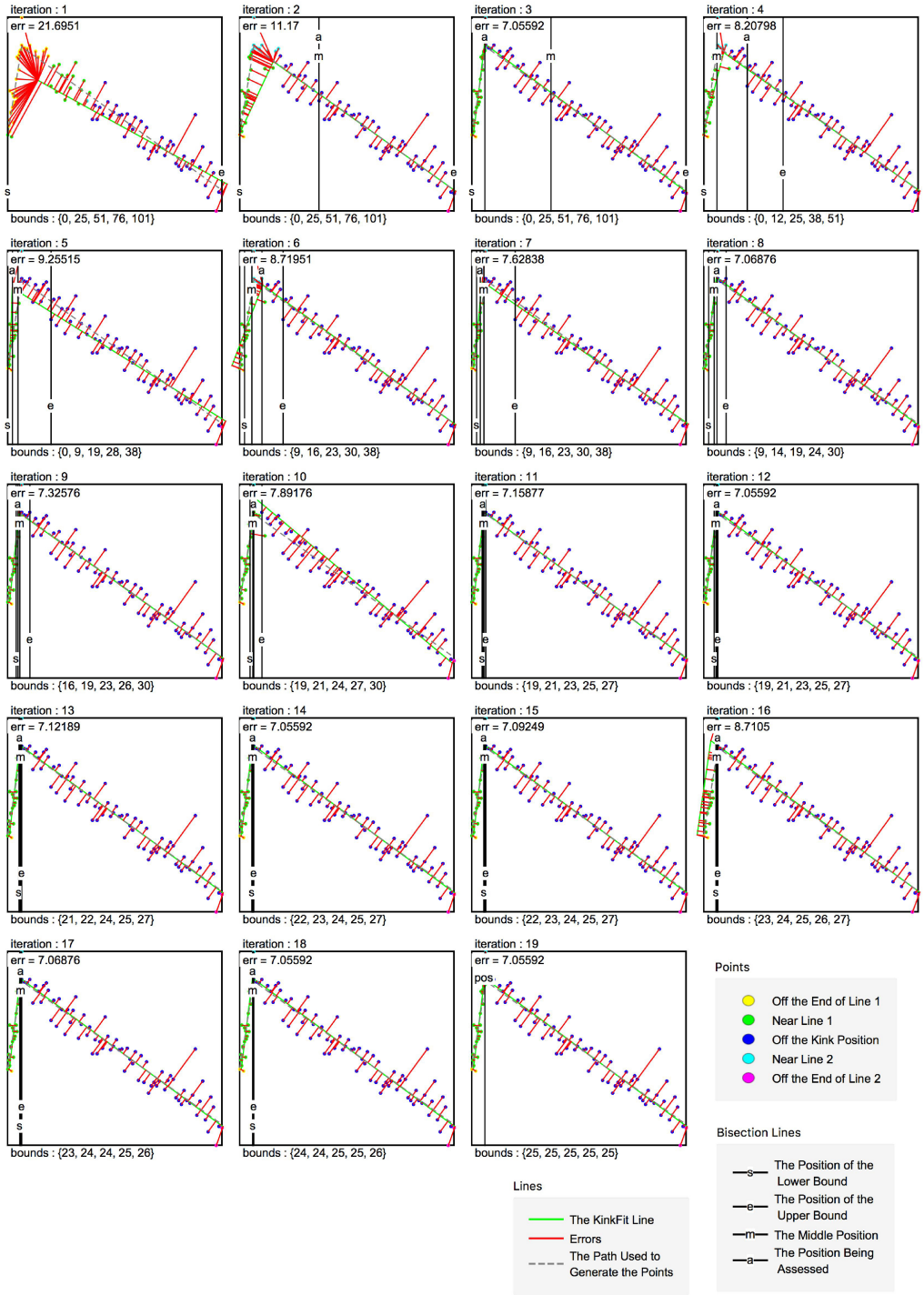


Fig. 4.25 Example A of the KinkFit algorithm in action, showcasing its reliability over several iterations.

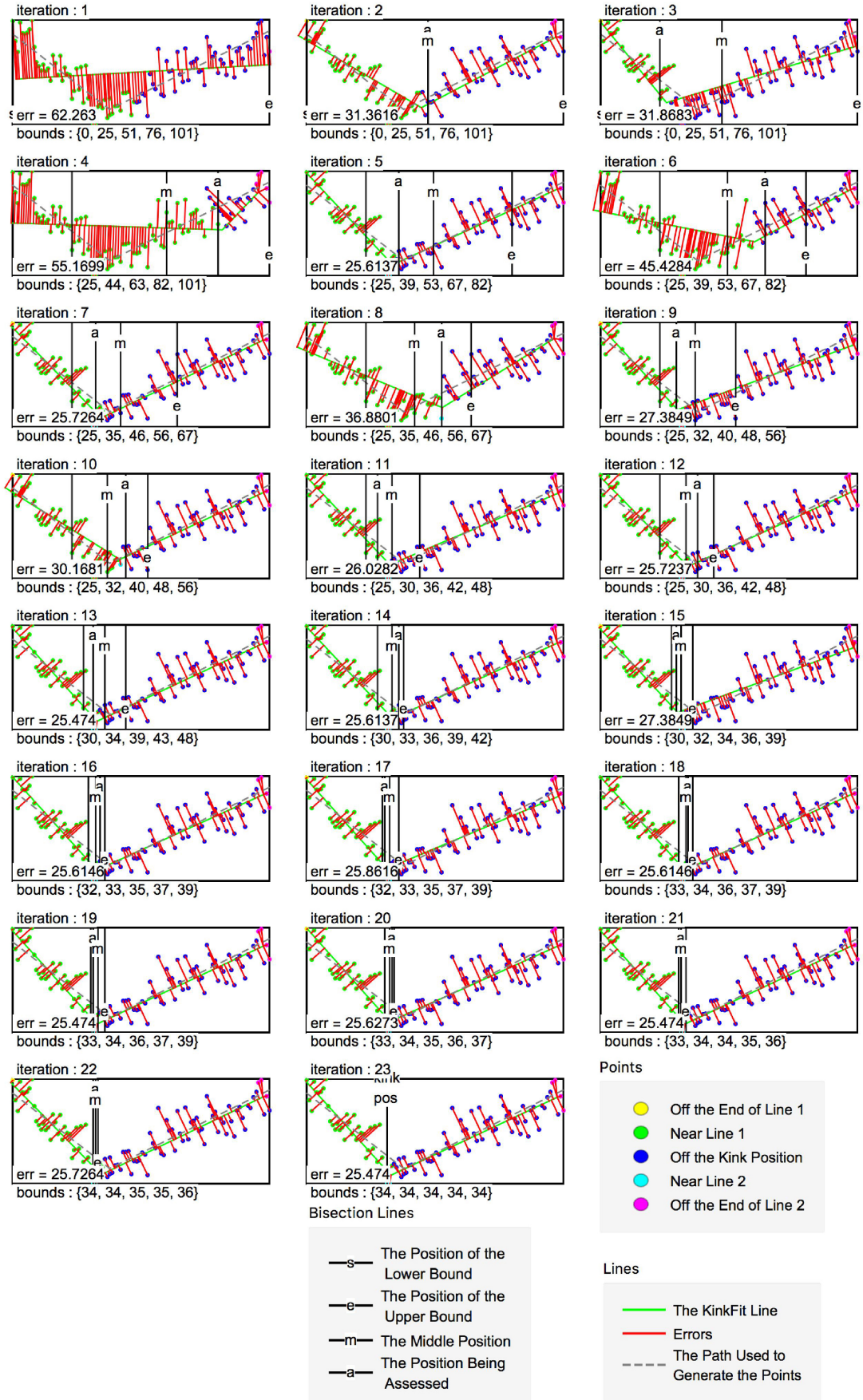


Fig. 4.26 Example B of the KinkFit algorithm in action, showcasing its reliability over several iterations.

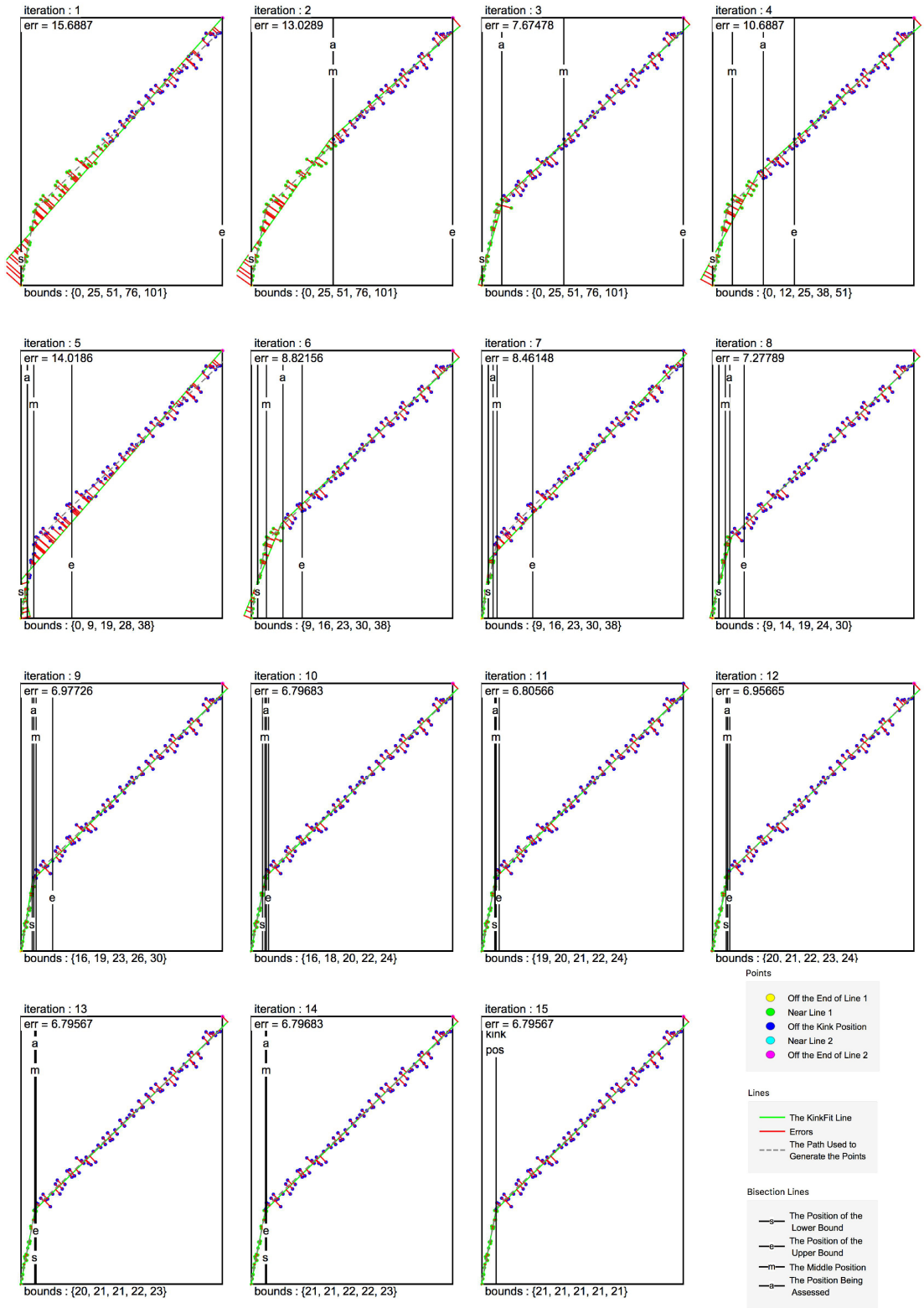


Fig. 4.27 Example C of the KinkFit algorithm in action, showcasing its reliability over several iterations.

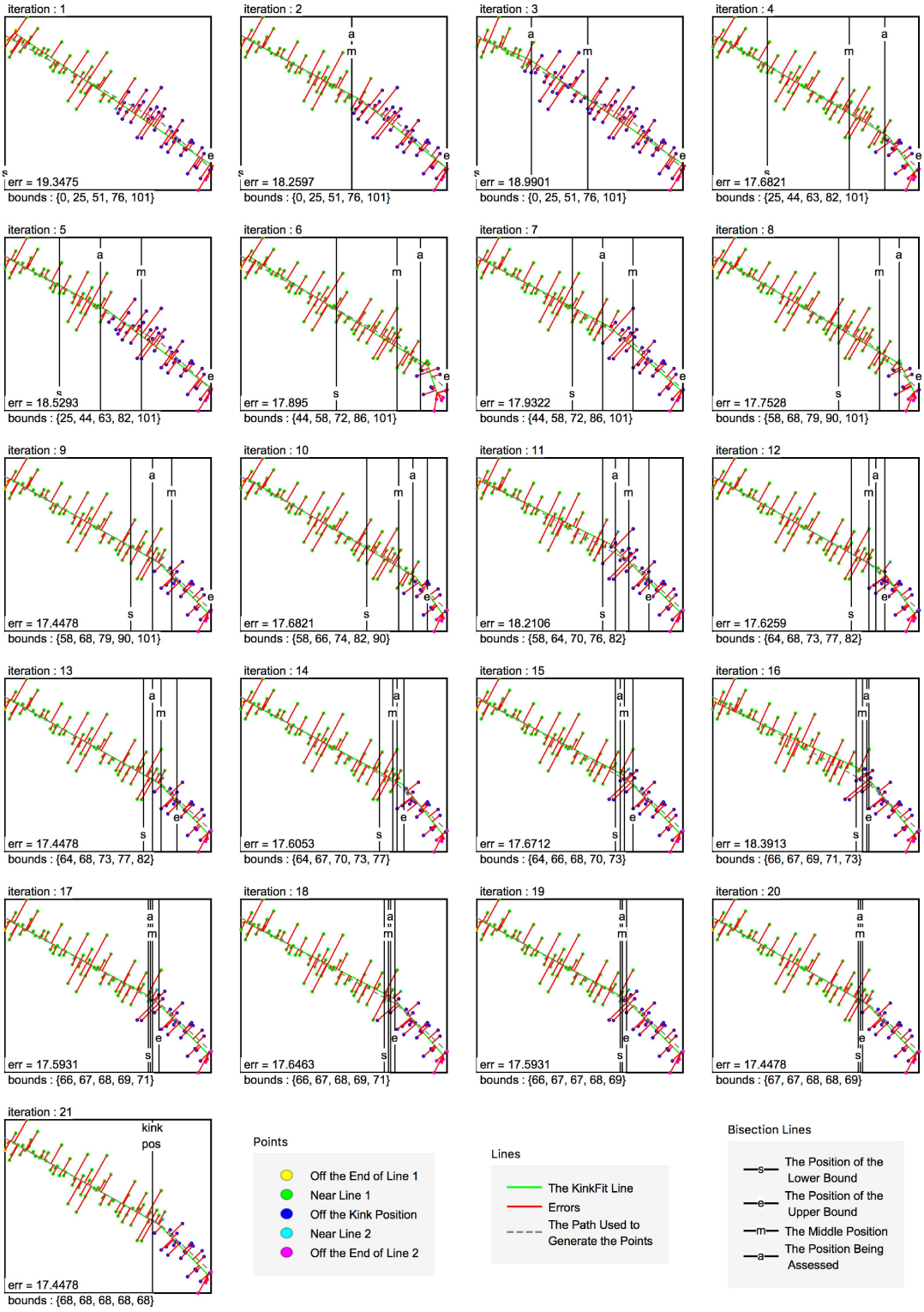


Fig. 4.28 Example D of the KinkFit algorithm in action, showcasing its reliability over several iterations.

Chapter 5

Future Work and Discussion

5.1 Results and Evaluation

The primary focus of this project is to develop and implement a color space algorithm that is efficient enough to run on mobile devices, but also accurate enough for broad applications. In Chapter 2, a color space conversion method was developed which simultaneously avoided using floating-point operations and preserved all the chromatic information in the source color space. Preserving all the chromatic information from the source color space is something which is not even achieved by floating-point methods because — even if we assume floating point numbers are represented with infinite precision — these methods only use the floating point representation internally and lose information when converting back to the integer data types for the rotated chromatic channels (see Section C).

5.1.1 Timing the Methods

To evaluate the integer-based color space conversion method in comparison with an equivalent floating point method, a floating point color space conversion was implemented which first converted the pixel value into a double precision unit range, performed the rotation using a double precision rotation matrix and then redistributed with the redistribution function described in Chapter 2 to the destination integer range. This is compared below with the optimized integer method presented in Chapter 2.

Obtaining clean running times for multi-core CPUs running multiple threads is problematic. To account for the variability in performance, the two algorithms were run using the same input 1,000 times. Theoretically, the time should be the same for every run with the same input, however, as can be seen in Figure 5.1, the timings vary away from a baseline

with a statistical positive error. It is reasonable to assume that this variation is due to interrupts from system processes, and so the algorithm performance is best measured by taking the minimum runtime from this set of 1,000 runs.

For the image used in this test, it can be seen that the integer method is about 4.9 times faster than the floating point method. This is a significant gain given that the color space transform is the most computationally-intensive operation in the entire algorithm.

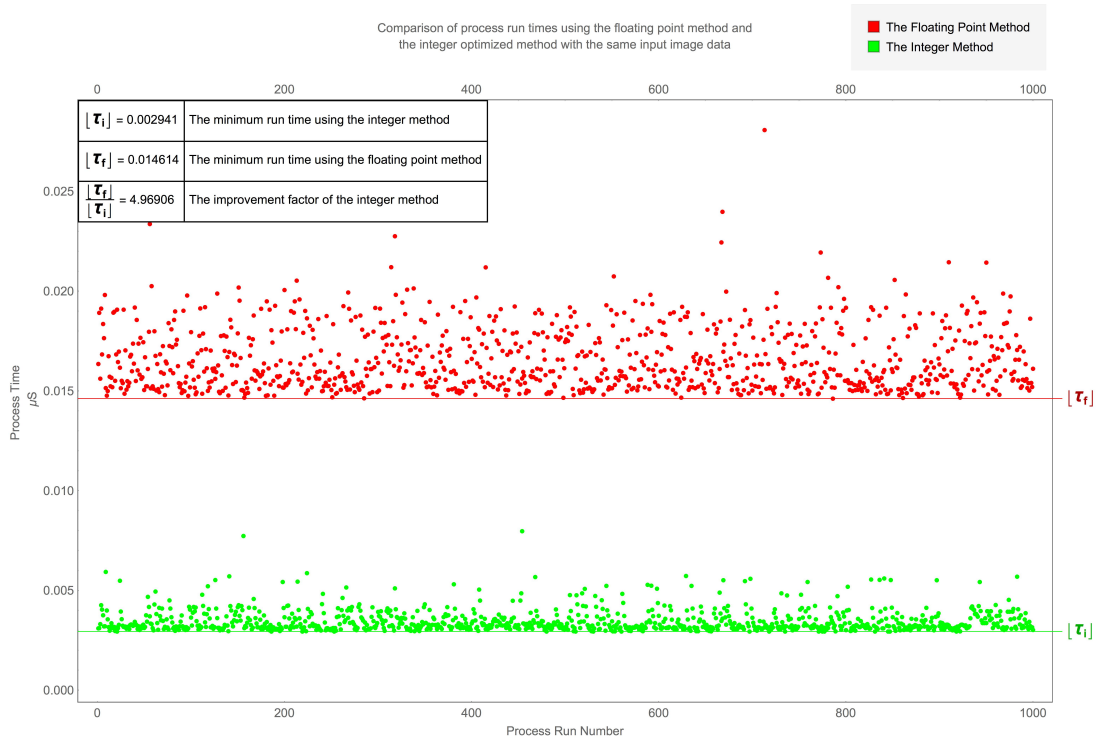


Fig. 5.1 Timing variation plot. Comparison of process run times using the floating point method and the integer optimized method with the same image input data.

5.1.2 Optimization Variation with Input

Now we evaluate the performance of the algorithm with differing input images. For the purpose of this test, images generated with varying tonal characteristics is done by generating images with random pixel values within random ranges. Each image was processed 1,000 times and the minimum runtime from those 1,000 runs was taken to be an indication of the difficulty of that image when presented to the two algorithms. The algorithms were run over 100 different images, and the mean and the standard deviation from that mean were found for each of the two methods. The performance of the algorithms can be seen in Figure 5.2 below.

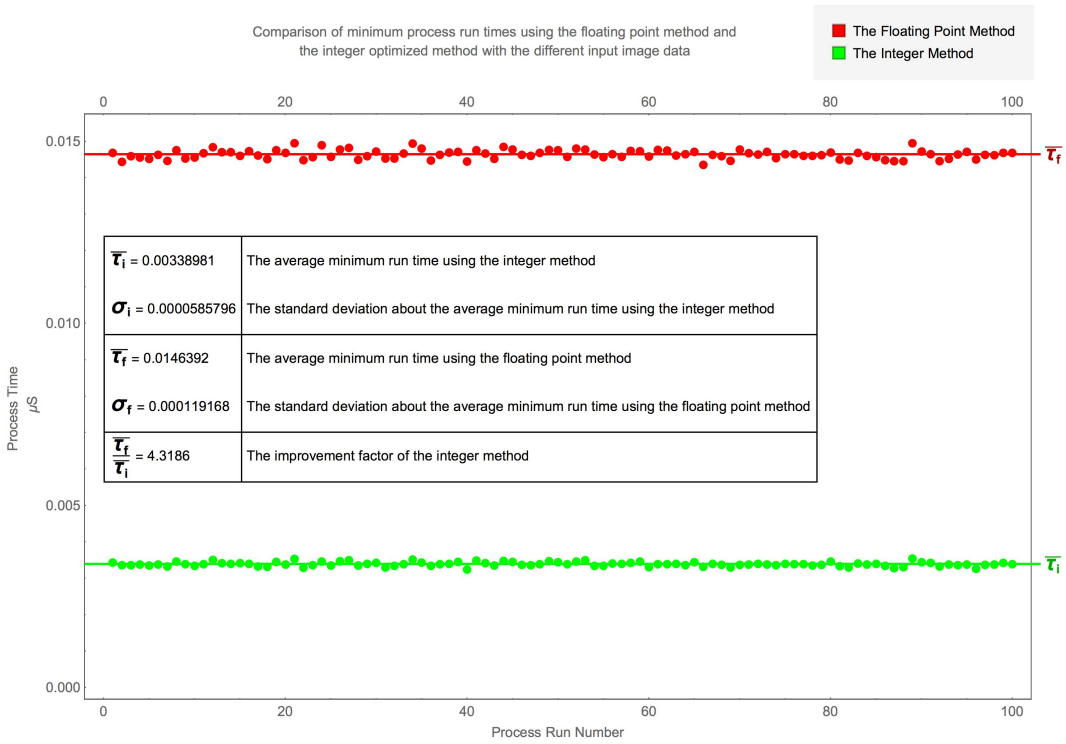


Fig. 5.2 Tone variation plot.

Here the performance gain can be seen to be slightly lower than for the test image used above. However, it can be confidently said that the method presented in Chapter 2 is 4-to-5 times faster than using the usual floating point methods.

5.2 Future Applications

5.2.1 Medical Applications

The benefit of using the color space conversion algorithm in medical applications is that it's lossless without negatively affecting performance. This is especially beneficial when performing comparisons between chromophores with very subtle differences such as oxy-hemoglobin and deoxyhemoglobin or the two types of eumelanin, as losing information makes telling them apart that much harder. These two benefits of the algorithm immediately suggest a number of possible future applications.

One possible application of the algorithm is monitoring inflammation. By adapting the color space to distinguish between healthy and inflamed tissue, one might be able measure the degree of inflammation over a period of time by, for the sake of example, photographing

parts of the user's body prone to inflammation throughout the day and using the algorithm to perform the comparison. This has the benefit of measuring the degree of inflammation without having to rely on subjective reporting, and could allow individuals to discover triggers for conditions like eczema, arthritis and other such inflammatory diseases. Combining this approach with diet and lifestyle monitoring functions — other factors which may contribute to inflammation — could serve as a simple and empowering means of increasing users' quality of life.

In mole mapping, a series of photographs of moles are taken over a period of time to track changes in size, color or shape in order to detect conditions such as malignant melanoma. However, identifying changes in such skin lesions is often limited to the doctor performing naked-eye comparisons between RGB photos of the areas of concern (Figure 5.4). This poses a problem when checking for changes in color, in part due to the RGB color space not separating the luminosity from the chromatic information and differences in lighting conditions. Using our color space, it is possible to highlight such changes more clearly; the color space can be adapted to detect changes in coloration between successive skin lesion images, thereby eliminating the need for guesswork. Additionally, by floating the mean the algorithm can account for variable lighting conditions.

Blood flow patterns in the retina are a good indication of eye health. As such, a retina imaging app for monitoring these blood flow patterns is another potential application of the algorithm. It is possible to adapt the color space to discriminate between oxygenated and deoxygenated blood flow and use the image alignment such that, with a simple lens manipulator which fits over the phone's camera, one could photograph their own retina daily and measure the difference overtime. The alignment algorithms could be used to monitor any burst blood vessels, even within the eyeball itself. Another, similar application would involve detecting for anemia; the skin under the eyelids lacks pigmentation, appearing pale if there is little blood. One might be able to take a photo of the skin under the eyelids with their phone camera and use the algorithm to check for a lack of blood flow.

As for applications beyond phone cameras, a number of different medical cameras have been designed for medical diagnostics (see Figure 5.5), especially for use in telemedicine and thermal imaging, many of which are commercially available. These cameras are typically HD video cameras, some of which support a variety of attachments for different types of examinations such as tongue depressors for laryngoscopy. However, these cameras also tend to use the RGB color space, which again has the disadvantage of requiring the examiner to perform a naked-eye comparison, relying on their knowledge and experience to make a diagnosis. By adapting our color space to target specific chromophores, it may be possible

to accurately diagnose any number of conditions without relying as much on expertise.

Fig. 5.3 DermLite DL1; A dermatoscope attachment available for several smart devices, used in diagnosing skin disorders.

Fig. 5.4 A skin lesion photographed under different lighting conditions. (Photos by Kevin Jakob, Illingworth Research.)

Fig. 5.5 A few different medical examination cameras.

5.2.2 Scientific Applications

5.2.2.1 A Study of Human Chromatic Diversity

In Chapter 3, we used a sample set of skin images from three individuals; it was interesting to see, in the lumiochromatic color space LCaCb, how close these skin statistics were to each other in the chromatic plane. It would also be interesting to perform the work of Chapter 3 for a large and chromatically diverse group of people. An attempt was made to look at a diverse sample of individuals using images collected in the Humanae project (**?**). This failed, however, as we had no control over the camera or the image compression performed before making them available on the Internet. Ultimately, it would be interesting to see if it were possible to estimate the levels of the skin chromophores identified in Chapter 1 using the skin statistics of the individual.

5.2.2.2 Chromatic Statistics for Injuries and Defects

If we have the normal skin statistics for an individual, we could take images of a site of injury and build the chromatic histogram (Figure 3.5). We could find the difference between the sets and create a profile for the injury or defect. In order to do this, it would be useful to have built the statistics for the injury site prior to injury; this is an issue if this is to be done in-lab as it would necessitate causing the injury or defect in order to study the effect. This could be overcome using crowdsourcing. For instance, one might go skydiving as a hobby and so would often bruise their legs, so we could build the skin statistics for their uninjured skin and simply wait for a bruise to appear.

An issue of this approach would be the consistency of lighting conditions; this could be overcome if we have adapted the method to allow us to set a new white point as outlined in Section 5.3.1. We could take an uninjured skin patch for which we already have a previous image from which we build the skin statistics, and comparing the two would allow us to find a white point which accounts for the ambient lighting conditions. Our generalized transform could then compensate for the changing lighting conditions. Any difference in the statistics now is due to the injury, and our theoretical model built in Chapter 1 should allow us to interpret this in terms of the chromophores, the key one being the bilirubin, which is a chromophore which causes the yellow-and-green coloration in bruises.

5.2.2.3 Repetitive Strain Study

The algorithm we developed in Chapter 4 focuses on the mechanical stress to the fingertip when pressing on a surface; in the Figures 4.9, 4.10 and 4.11, it can be seen that the stress also shows in the knuckle. In fact, the mechanical strain can be seen in all the joints of the hand. This suggests that we could build an algorithm which monitors the stress in the joints of the hand by observing the blood flow using the routines developed above. By monitoring the strain put on the joints of an individual's hands while they perform a task, we could measure the overall strain that they have placed on their finger joints. If we combine this with the techniques to measure inflammation presented in Section 5.2.1, we could get a measure of the damage suffered by the individual.

Once developed, such a monitoring system could be used in studies which assess different ergonomic designs and workplace practices.

5.2.3 Computer Vision Applications

5.2.3.1 A Novel Canny Edge Detection Algorithm

The trinary classification developed in Chapter 4 can also be used for an edge detection routine which operates similarly to the Canny edge detection. The Canny edge detection operates by finding edges using a high threshold which is then lowered as the routine traces along the edge, allowing the edge to be extended from a strong edge into a weak edge. This also allows Canny edge detection to "hurdle" over minor defects in the image of the edge. The trinary classified image can be binarized and presented to standard edge detection techniques, thus we can find all of the strong edges.

The edges which end surrounded by possible "probably not skin" values can easily be found and marked. Other marked ends which lie close to each other are joined if the pixel values on the path between them are all "probably not skin" (i.e. "1") or higher. This is similar to the Canny edge detection in that it allows us to join edges by using a lower threshold across the joint. It differs from Canny, however, in that it does not allow the edge to extend into a lower threshold edge. This would be undesirable using the trinary classification because the low threshold "probably not skin" classification often appears in patches which extend across the true edge often resulting from shadow or reflection. This method could be refined by taking into account the orientation of the edges at their end points, which could be used to determine how edges should actually be connected. This is similar to the "hurdle" method presented in Section E.0.1, but applied to edges.

5.2.3.2 Finger Feature Detection

The modeling techniques developed in Chapter 4 can be relatively straightforwardly extended to model the entire digit, or even the whole hand. In Chapter 4, we found that the distal width could be used to find a good estimate for the position of the knuckle using the distal, middle and proximal ratios. With this initial model, the next question is what features can we expect to find at these positions in the image? The luminosity axis shows a striking feature at the knuckle position: the knuckle wrinkles. These wrinkles are consistent to each knuckle (i.e. when flexed, they present in the same way), and they're easily identified by orientation, running across the digit as opposed to along it. Additionally, the knuckle wrinkles curve around the center of the knuckle. This could be used to refine the model and track the knuckles. In the chromatic channels, they also display blood flow when flexed.

5.3 Color Space Algorithm Improvements

Here we suggest several improvements which could be made to the color space algorithm developed in Chapter 2. The improvements presented herein address generalizing the algorithm to accommodate the characteristics of cameras other than the iPhone's. We consider three adaptations; one to allow for multi-spectral cameras, another to allow for cameras which do not have a fixed white point, and a third to handle RAW images captured from the CCD without the iPhone's pre-processing. In practice, these three adaptations need to be combined. Hopefully it shouldn't be too difficult to imagine how this could be done, but in terms of practically proceeding, each one would need to be tackled individually first.

5.3.1 Generalizing the Rotation

In Chapter 2, we derived an expression for a rotation matrix consisting entirely of integers which rotates the RGB pixel values into a custom color space with a luminosity axis and two chromatic axes. The orientation of the chromatic axes is specified using a free rotation θ about the luminosity axis. The only restriction on the free parameter θ is that which results from the requirement for the matrix to consist entirely of integer values within a specific range. The question remains: is it possible to achieve a similar result without restricting one of the axes?

In this work, we assumed that an 8-bit per-channel value of $(255, 255, 255)$ corresponds with white. This, however is a result of the pre-processing on the iPhone. Extending this work to other devices or accessing the iPhone's RAW camera feed, it may be desirable to

use a different point for the camera's white point. Another reason is to adjust for local ambient light conditions; it is common in digital photography to choose a pixel value from a reference white object to set a white point for the image, allowing a color correction to be performed.

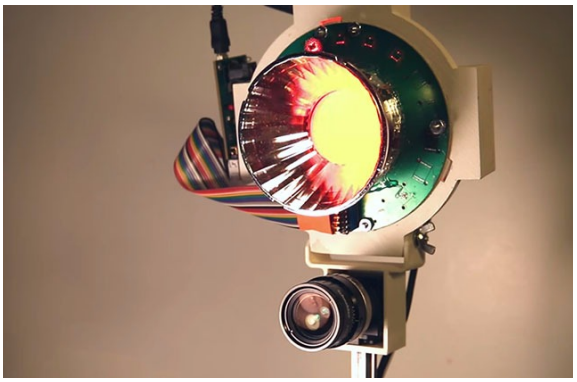
An arbitrary rotation is determined by three angles, and so it's conceivable that the work of Chapter 2 (Section 2.1) could be repeated for this arbitrary rotation. Whilst it may be possible, an alternative approach may be preferable given the complexity of the general rotation equations. First we assume that the solution exists, then form an initial guess at the solution by quantizing the floating point representation of the matrix. Next, we find the maximum error produced by our approximate quantized rotation by rotating the corners of the RGB cube and comparing the results, then numerically search around this approximation for improvements. The question that remains is what search area should be included.

Because the general rotation matrix can be split into the product of three individual matrices, each of which is dependent on only one of the three free parameters, the work in Section 2.1 could be repeated relatively straightforwardly for each of these three individual matrices, and from this — for a given quantization criteria — a minimum and a maximum angle step size (Figure B.1). If we have for the separate matrices a minimum and a maximum step size, it's a reasonable idea to search around our initial approximation in steps of the minimum step size to either side out to a maximum distance of the maximum step size. For each of the three angles, we search within a cube with sides defined by the maximum step sizes for each of those rotation matrices on a cubic grid defined by their minimum step sizes. The result of this is that there is a relatively small set of possible optimized choices. Although this is not solving the problem to the same degree that was done in 2.1, where we reduced it to a mathematical function, it provides a practical way of achieving the same results without excessive computational effort in the setup.

5.3.2 Multi-Channel Color Spaces

As mentioned in Chapter 1, the RGB color space is an approximation to the full spectrum of light. However, cameras which are not confined to the three-channel approximation are becoming commercially available (Figure 5.6); such a multi-channel image contains far more information about the objects in frame. So where the camera is being used for computer vision tasks where the objective is not to produce a pretty picture to present to the human eye, this allows the chromatic space to discriminate between different objects and surfaces in ways which are unseen by the standard RGB camera point of view.

(a) Triwave EC701.



(b) HyperCam by University of Washington and Microsoft Research.

(c) Optec multi-spectral camera.

Fig. 5.6 A variety of multi-spectral cameras currently in use.

The basic design of these multi-spectral cameras is essentially the same as for the RGB cameras where the spectrum of light on each point in the scene (roughly speaking; see Chapter 1) is represented by a combination of Gaussians, the only difference being we have more Gaussians. Each channel can therefore be considered an orthogonal axis in a multi-dimensional color space. The multi-dimensional color space contains a point which corresponds to white, so it is just as possible to construct a color space with a luminosity axis and $n-1$ chromatic axes for an n-channel image. Transformation to this color space is defined by a multi-dimensional rotation; this rotation could be optimized in a similar way to that presented in Chapter 2. However, our requirement in Chapter 2 was that the immediate result of the rotation would fit into a data type no more than twice the bit depth of each channel. An n-channel color space will be rotated by an n x n rotation matrix; if we have m bits per channel and the quantized rotation matrix is expressed in l-bit integers, the result is $l+m+(n-1) \leqslant 2m$. For the larger values of n, the rotation matrix would have to be expressed in smaller data types, so aside from special rotation angles which just happen to produce integer rotation matrices, it's unlikely that our criterion can be met. So for multi-channel images to still be able to use integer rotation matrices, the target data type will undoubtedly have to be significantly larger.

5.3.3 Using the RAW Image From the Camera

The RAW camera image contains all the information captured by the camera, but without any pre-processing. Individual CCDs have different sensitivities in each of their channels, and so the corresponding bit depth of each channel may actually vary. This is likely to be the case for the iPhone's camera as we found in Chapter 3 that certain RGB values appear to be inaccessible. These inaccessible values also appear to be relatively evenly-spread throughout the RGB cube, which produced what we referred to as "speckling" in the statistics gathered in Chapter 3 as seen in Figure 3.10.

A possible explanation for this is that one or more of the iPhone camera's channels are not truly captured at an 8-bit depth, but a smaller range of values which the CCD actually captures is stretched to fit the full 8-bit range. It is also entirely possible that one or more of the channels may actually be captured at a higher bit depth than 8-bit. But in terms of the algorithm, we now need to allow each channel in the source to have a variable bit depth. This will affect several things; it changes the integer range of allowed values in a rotation matrix and the algorithm for determining the optimal angle could be adapted by weighting the channels appropriately. But otherwise, adapting to a RAW camera image should be

relatively straightforward.

5.4 Implementation Improvements

5.4.1 An Empirical WoBo Algorithm

The White-out Black-out algorithm outlined in Section 3.2.3 uses theoretical values based on the extent of the data type used to store the RGB image. Looking at the empirical results presented in Figure 3.1, there's more to the WoBo behavior than simply saturating the data type; the performance of the CCD changes with overall ambient light, and the pre-processing performed on the device also has an effect which takes the pixel values away from the naive data point values using the data type extent.

Although it may be possible to reverse-engineer both the pre-processing and CCD characteristics simultaneously, this would be a difficult task at best. However, if for a device we were able to obtain the RAW image from the CCD, then repeatedly performing empirical tests, such as those performed to generate Figure 3.1, the CCD characteristics could be obtained. If we know how the CCD detects objects of a certain color under different ambient luminosities, then a more nuanced algorithm can be used to determine if a certain pixel value has suffered from white-out or black-out.

5.4.2 Auto-Adjusting the Variance

The algorithm presented in Chapter 4 uses a reference image to float the mean, yet the algorithm makes no attempt to adjust the variance to local ambient light conditions. This is found to work well, however it was initially found that using a standard deviation 2.3 times larger than the value found in Chapter 3 produced better results= (see Figure D.2). Although the algorithm was tested against a selection of background colors, backgrounds which are notably close to skin colors were avoided. For practical applications, it may therefore be useful to use a value for the standard deviation close to that used in Chapter 3.

The images chosen to be included in this document were taken against a green background because that gave the cleanest results; a value of 2.3σ produced good results in this ideal background condition, so we'll choose 2.3 as the overall upper limit and 1σ as the overall lower limit. We have a reference image where we have a good idea that there is a finger, and a portion of that finger is within a known frame, so the lower criteria is that nearly all, if not all, the pixel values within that frame are categorized as skin using the new

value for the standard deviation, so for a choice between 1σ and 2.3σ, we can "score" the choice as follows:

1. The proportion of pixels within the known skin region successfully classified as skin.
2. The number of edge points found using the Filament fill algorithm which are classified as bad edge points which also lie outside the approximated digit edge.
3. The number of edge points found by Filament fill which are classified as bad edge points, but which lie within the edges of the digit.

So, a score where measure 1 is too low suggests that σ should be increased, as pixel values which we know to be skin are classified as not skin; if measure 2 is too high, then we need to reduce the value of σ; and if measure 3 is too high, then we need to increase the value of σ.

This suggests a way that an algorithm could be developed which appropriately adjusts the value of σ striking a compromise between false negatives and false positives. The actual values, thresholds and criteria used can best be determined empirically.

One way of proceeding may be to combine the three measures by taking the weighted product of measures 2 and 3 with the weighted reciprocal of measure 1. We combine the measures in this fashion and minimize the result with respect to σ.

5.4.3 Statistics Gathering Method for the App

Given that, in the implementation, we have "floated" the mean and "relaxed" the standard deviation, and a method is suggested for adaptively adjusting the standard deviation above, it is natural to consider whether the app can handle all the work from Chapter 3. Collecting the statistical model following Chapter 3 is entirely possible, as at no point is any human intervention necessary. However, the Chapter 3 methodology requires that a set of images are captured against a uniform, highly chromatically-contrasting background.

An interesting possibility is whether the work of Chapter 3 can be somewhat replicated using the same reference image which is used by the app to float the mean and build the initial fingertip model. The difficulties with this idea are the same two difficulties faced by most statistics-gathering methods; sample size and selection bias. To address the first difficulty, we've seen that the number of pixels within the frame which correspond to skin is sufficiently large to usefully adjust the mean of the distribution. This suggests that the sample size should not be a significant problem. Selection bias, on the other hand, comes

from asking the user to select a portion of their finger which they consider to be most skin-like; this is not an explicit request of the user, but an implicit request, i.e. when asking a person to supply a sample of skin, they will supply the most skin-like sample they can. This selection bias will likely exclude the edges of the digit, strong features and poorly-lit portions of the skin. As a result, while the mean almost certainly will be fine, the variance will be narrow. So, it's easy to imagine using techniques suggested for auto-adjusting the variance (Section 5.4.2) to make the distribution more inclusive.

One final refinement suggests itself at this point; the resulting color space can be used to model the fingertip and mask the image such that only on-digit pixels are present. In doing this, we now have a broader sample set of pixel values which don't suffer from selection bias. These pixel values could be passed back through the statistical model algorithm to produce a skin chromatic model which is less susceptible to human input.

5.4.4 Improved ICWaS Alignment

To align successive frames, the ICWaS developed in Chapter 4 uses OpenCV's template aligning routines which operates by performing a translation of one frame and computing a per-pixel difference. This works because the alignment is already quite good. There are two ways in which this can be improved; the allowed transforms can be increased to include rotation and perspective adjustments, and the alignment methodology can be improved to utilize more constant features of the fingertip image than the per-pixel approach.

It's a simple matter to allow for a more general transform, however the per-pixel based comparison is wholly inappropriate to find a more general transform as we would have to try all possible results of the transform and compare them. This would be computationally far too expensive. So, to find a more general transform, we must turn each frame into a set of feature points which can be put into a correspondence, or perform some image transform which simplifies the alignment problem.

5.4.5 More Designed Metric for Mechanical Stress

The metric method which is very briefly presented at the end of Chapter 4 (see Figures 4.9, 4.10 and 4.11) relies on a pixel-to-pixel comparison; it is well known that pixel-based methods are unreliable for detecting movement, and our mechanical stress measurement relies on detecting the flow of blood. The metric should therefore account for neighboring pixels.

There are many possible methodologies for detecting movement which would improve

the pixel-based method of Chapter 4. Mechanical stress tends to move pooled blood from one area to another within the stressed tissue. The upshot of this is that as one portion of the image loses blood, another gains it. This is different to the circulatory blood, which is pumped in through the arteries and capillaries and flows out through the veins. A better metric may be to compare blobs in the negative and positive chromatic difference images. To be clear, the algorithm would find blobs using a negative thresholding and a positive thresholding; the largest blobs in each of these spaces would be considered good indicators of pooled blood flow.

Another possibility arises from incidental observations of the pooled blood flow images; for each individual's digit, the pooled blood flow pattern is fairly consistent under similar mechanical stress conditions. This suggests that it may be possible to construct a blood flow model for a given digit. Regarding the app, we can imagine generalizing the reference image idea to asking the user to press on a surface several times, allowing the blood flow model to be generated for that digit.

5.5 Conclusion

The finger press algorithm is a simple application to demonstrate the viability of detecting blood movement using a standard camera on a mobile device. Many previous authors have dismissed using bespoke color spaces for such applications simply because the color space transform itself is computationally intensive and significantly loses information when applied using standard library color spaces.

It is hoped that, with the rigorous optimization of the transform itself — which is considered in great and incredibly tedious detail in Chapter 2 — that these color spaces can see greater application in the future, although it's recognized that the level of attention that's currently required to make the routine as efficient at the one designed herein will likely be beyond the patience of many practitioners, some of whom are medical professionals and not computer scientists. Also, it is hoped that the work presented here has addressed the concerns about the associated loss of information.

Finally, the use of a 2D Gaussian model for the skin color space has been the source of much debate; some authors insist that the statistical model be extended to all three dimensions (**?**), but hopefully it can be seen from this work that the luminosity channel can be effectively removed from consideration because its influence on the statistics is an artifact due to white-out and black-out. In fairness to the authors who have argued for its inclusion in the model, the luminosity is considered by the Quaternary Pixel Classification

outlined in Section D.2.1, however this part of the algorithm is theoretically derived from the two-dimensional chromatic statistics. To be clear, the authors who argued for the full three-dimensional model were correct in that luminosity has an effect, however they were including in the skin model camera effects which can be compensated for by application of theory.

The secondary controversy is the computational cost of applying a 2D Gaussian, which has led to authors using elliptical functions, rectangular functions, etc. The algorithm developed in Chapter 2 takes advantage of all these methods to produce a composite algorithm which very efficiently applies a 2D Gaussian distribution. The central trick employed was to use a bespoke color space in which the long axis of the 2D Gaussian is aligned parallel to one of the axes of the color space. This means that the 2D Gaussian is functionally expressed as the product of one-dimensional Gaussians. The advantages of using a three-dimensional model would appear to be re-gained using the Quaternary Pixel Classification algorithm, and by judicious collection of the skin statistics as seen in Chapter 3.

In conclusion, we have presented a rigorous statistical method and model for human skin detection which has, to my understanding, addressed all the concerns expressed in the current literature.

Appendix A

Chapter 2 Mathematical Results

A.0.1 Rotation Matrix

Any rotation about an axis can be represented by a 3x3 square matrix in a 3D space. Since they are invertible, they're guaranteed to be non-singular. However, as there are many ways in which to rotate an object from one position to another, or use a combination of different rotations to get to the same point, they aren't necessarily unique. For this application, we require rotation about three different axes, which can be expressed thus:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{A.1}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{A.2}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

Any rotation which orients the RGB color space such that one of the new axes lies along the luminosity direction is sufficient; a rotation which aligns the L axis along the luminosity direction is produced by a rotation of $\frac{\pi}{4}$ about the red (**R**) axis, followed by a rotation of $\arctan\frac{1}{\sqrt{2}}$ about the green (**G**) axis. This leaves one free rotational degree of freedom about

the L axis. The resulting rotation matrix is given by:

$$\mathbf{R}_{xyz}(\theta) = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\sqrt{\frac{2}{3}}\sin\left(\theta+\frac{\pi}{6}\right) & \sqrt{\frac{2}{3}}\cos(\theta) & -\sqrt{\frac{2}{3}}\sin\left(\frac{\pi}{6}-\theta\right) \\ -\sqrt{\frac{2}{3}}\cos\left(\theta+\frac{\pi}{6}\right) & -\sqrt{\frac{2}{3}}\sin(\theta) & \sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-\theta\right) \end{pmatrix} \tag{A.4}$$

Where θ is the remaining rotational degree of freedom.

Using the standard rotation matrices, we get a luminosity axis which spans the range $0:\sqrt{3}$. However, the length of the two remaining axes are dependent on the value of θ used. This is a problem because, ultimately, we want the axes to fit in a range of an appropriate data type. It would be more useful to have a matrix which provided the specified rotation and scaled the axis to known lengths. In the case of the luminosity, this is straightforward; simply divide by $\sqrt{3}$. In the case of the other two axes, we need an explicit form for the lengths of the axis resulting from the rotation.

Because the absolute values of the axes in the color space have no meaning, we're only interested in the position along the axis relative to its start and end, equivalent to talking about the position in the axis relative to $0:1$, compared to about $0:255$ in unsigned, 8-bit integers. The upside is that if we're rotating the cube about its corner, we're interested in the minimum and maximum values possible along the new axis direction, which will correspond to a corner of the RGB cube. With the L axis aligned along the luminosity direction, the range of the L axis is 0 to $\sqrt{3}$. The x and y axes are symmetrical, spanning a range centered on 0. The range of their values is dependent upon the remaining degree of freedom.

We need to know, in each of the axes, how far out each point is. Because we're effectively rotating a hexagon, whatever the answer is, we know the function is going to be periodic, repeating every $\frac{\pi}{3}$ radians, so we only have to solve it in the $0:\frac{\pi}{3}$ region and then generalize. First we take the coordinates of the RGB cube and perform the rotation to find the values in the new color space.

$$\mathbf{R}_{xyz}(\theta)\cdot\begin{pmatrix}0&1&1&0&0&0&1&1\\0&0&1&1&1&0&0&1\\0&0&0&0&1&1&1&1\end{pmatrix}=$$

$$\begin{pmatrix}0&\frac{1}{\sqrt{3}}&\frac{2}{\sqrt{3}}&\frac{1}{\sqrt{3}}&\frac{2}{\sqrt{3}}&\frac{1}{\sqrt{3}}&\frac{2}{\sqrt{3}}&\sqrt{3}\\0&-\sqrt{\frac{2}{3}}\sin\left(\theta+\frac{\pi}{6}\right)&\sqrt{\frac{2}{3}}\sin\left(\frac{\pi}{6}-\theta\right)&\sqrt{\frac{2}{3}}\cos(\theta)&\sqrt{\frac{2}{3}}\sin\left(\theta+\frac{\pi}{6}\right)&-\sqrt{\frac{2}{3}}\sin\left(\frac{\pi}{6}-\theta\right)&-\sqrt{\frac{2}{3}}\cos(\theta)&0\\0&-\sqrt{\frac{2}{3}}\cos\left(\theta+\frac{\pi}{6}\right)&-\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-\theta\right)&-\sqrt{\frac{2}{3}}\sin(\theta)&\sqrt{\frac{2}{3}}\cos\left(\theta+\frac{\pi}{6}\right)&\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-\theta\right)&\sqrt{\frac{2}{3}}\sin(\theta)&0\end{pmatrix} \tag{A.5}$$

The extent of the new axis is found by taking the maximum and minimum values of each row, i.e. the extreme corner positions relative to each new axis. An additional symmetry of the hexagonal projection of the RGB cube allows us to say that — whatever functional form is taken by one of the θ dependant ranges — the other can be found by a simple phase shift. So, recognizing that the minimum value is simply -1 times the maximum, we have simplified the problem to solving:

$$\max\left(\pm\frac{\sin(\theta)}{\sqrt{6}}\pm\frac{\cos(\theta)}{\sqrt{2}},\pm\sqrt{\frac{2}{3}}\sin(\theta)\right)\quad\text{Where}\quad 0\le\theta\le\frac{\pi}{3} \tag{A.6}$$

A graphical representation of the problem can be seen in Figure A.1.

In the range $0:\frac{\pi}{3}$, both sin and cos are positive, therefore the axis ranges are given by the following:

	Min	Max
L	0	$\sqrt{3}$
Ca	$-\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-\left(\left(\theta-\frac{\pi}{6}\right)\bmod\frac{\pi}{3}\right)\right)$	$\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-\left(\left(\theta-\frac{\pi}{6}\right)\bmod\frac{\pi}{3}\right)\right)$
Cb	$-\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-(\theta\bmod\frac{\pi}{3})\right)$	$\sqrt{\frac{2}{3}}\cos\left(\frac{\pi}{6}-(\theta\bmod\frac{\pi}{3})\right)$

The lengths of the axis after rotation are given by:

$$\mathbf{L}(\theta)=\begin{pmatrix}\sqrt{3}\\\sqrt{\frac{2}{3}}\sin\left(\widetilde{\vartheta}\right)+\sqrt{2}\cos\left(\widetilde{\vartheta}\right)\\\sqrt{\frac{2}{3}}\sin\left(\widetilde{\theta}\right)+\sqrt{2}\cos\left(\widetilde{\theta}\right)\end{pmatrix}\quad\text{where}\quad\begin{matrix}\widetilde{\vartheta}=\left(\theta-\frac{\pi}{6}\right)\bmod\frac{\pi}{3}\\\widetilde{\theta}=\theta\bmod\frac{\pi}{3}\end{matrix} \tag{A.7}$$

It is convenient to produce a transformation which will result in axes of known length. Because the rotation cannot include a translation, we desire a transformation matrix which will result in the ranges $0:1$, $-\frac{1}{2}:\frac{1}{2}$, and $-\frac{1}{2}:\frac{1}{2}$. Such a transformation is easily obtained by

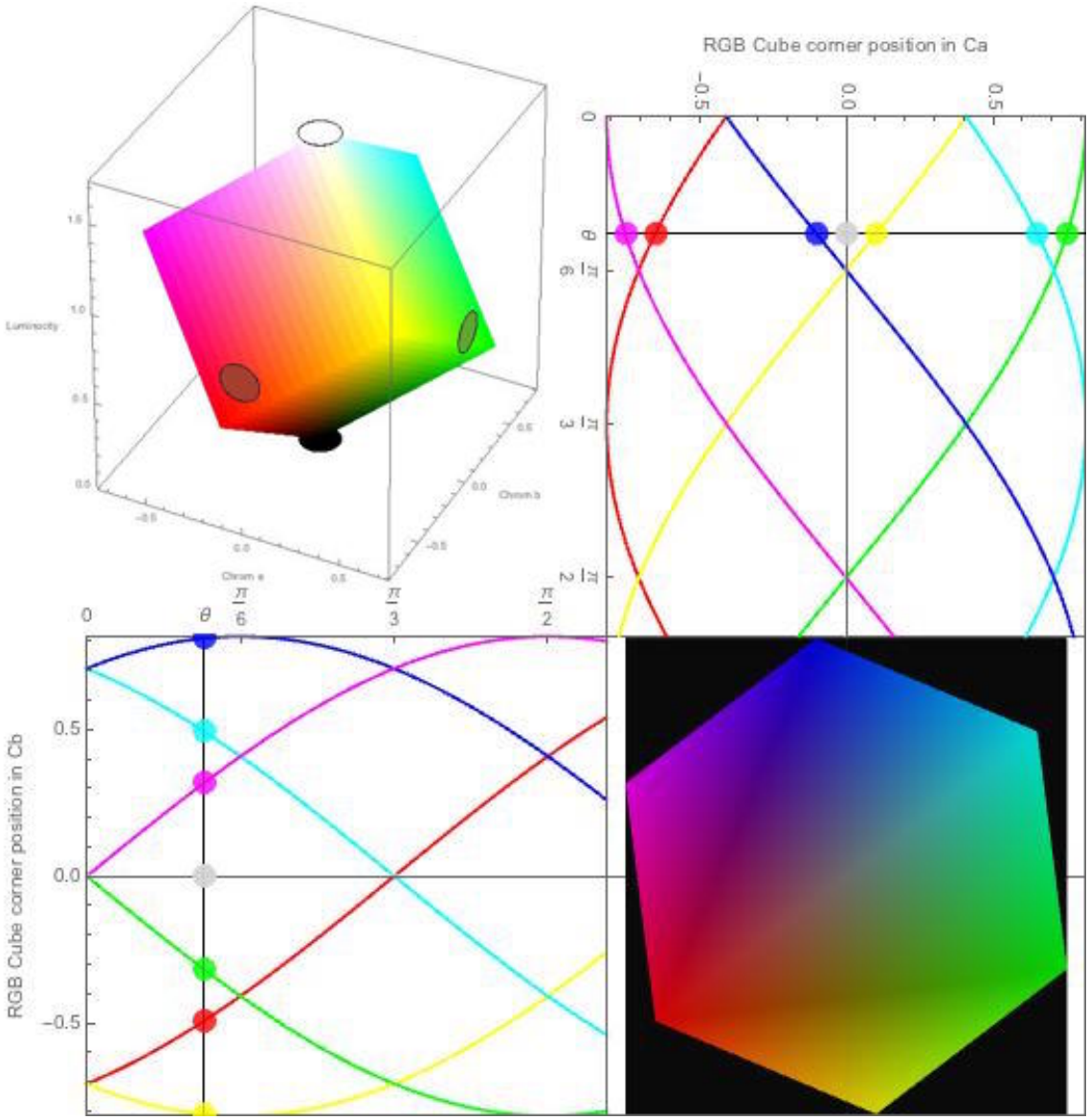


Fig. A.1 Evaluation of the color space problem. The cube in the top-left shows the positions of the axes in the rotated space; the white and black disks on the vertical axis represent the white point and black point of the luminosity axis, and the other disks represent the ends of the chromatic axes **Ca** and **Cb**. The graphs in the top-right and bottom-left show the positions of the corners of the RGB cube relative to the chromatic axes **Ca** and **Cb**. The bottom-right graphic shows the RGB cube viewed down the luminosity axis. The graphics were generated by interactive code written in Mathematica. The value of θ was an arbitrary value from a snapshot taken from the interactive graphic.

multiplying the rotation matrix by a diagonal matrix with the reciprocal of the maximums found above placed along the diagonal. This will scale each axis to a unit length.

The normalized 'rotation' matrix is given by:

$$\vec{L^{-1}}(\theta) = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{\frac{2}{3}}\sin(\widetilde{\vartheta})+\sqrt{2}\cos(\widetilde{\vartheta})} \\ \frac{1}{\sqrt{\frac{2}{3}}\sin(\widetilde{\theta})+\sqrt{2}\cos(\widetilde{\theta})} \end{pmatrix} \quad \text{where} \quad \begin{matrix} \widetilde{\vartheta} = (\theta - \frac{\pi}{6}) \bmod \frac{\pi}{3} \\ \widetilde{\theta} = \theta \bmod \frac{\pi}{3} \end{matrix} \tag{A.8}$$

$$\overline{\mathbf{R}}_{xyz}(\theta) = \vec{L^{-1}}(\theta) \otimes R_{xyz}(\theta) \tag{A.9}$$

$$= \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{2}\frac{\sin(\theta+\frac{\pi}{6})}{\cos(\widetilde{\vartheta}-\frac{\pi}{6})} & \frac{1}{2}\frac{\cos(\theta)}{\cos(\widetilde{\vartheta}-\frac{\pi}{6})} & \frac{1}{2}\frac{\sin(\theta-\frac{\pi}{6})}{\cos(\widetilde{\vartheta}-\frac{\pi}{6})} \\ -\frac{1}{2}\frac{\cos(\theta+\frac{\pi}{6})}{\cos(\widetilde{\theta}-\frac{\pi}{6})} & -\frac{1}{2}\frac{\sin(\theta)}{\cos(\widetilde{\theta}-\frac{\pi}{6})} & \frac{1}{2}\frac{\cos(\theta-\frac{\pi}{6})}{\cos(\widetilde{\theta}-\frac{\pi}{6})} \end{pmatrix} \tag{A.10}$$

This matrix is no longer technically a rotation matrix as its inverse is no longer equal to its transpose. It now equal to:

$$\overline{\mathbf{R}}^{-1}_{xyz}(\theta) = \left(\vec{L}(\theta) \bigotimes R_{xyz}(\theta)\right)^T \tag{A.11}$$

A.1 Factoring the Rotation

The rotation matrix A.4 can be factored by rows to give a matrix which has one as the largest element in each row. This factorisation is used to facilitate the quantisation of the rotation matrix in Chapter 2.

The first row is simply factored by $\frac{1}{\sqrt{3}}$ to give all ones which fits our criteria. The second and third rows each sum to zero ($\text{uR} \cdot \mathbf{1} = [\sqrt{3}, 0, 0]$, where $\mathbf{1}$ is a vector of ones), allowing us to state that each of the rows has one element of the opposite sign to the other two elements. The discretization for the second and third rows then only requires rRange = 2 tRange. It is also possible to pull out common factors from the rows, resulting in all elements taking values between -1 and 1.

$$\text{uR}(\theta) = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \sqrt{\frac{2}{3}} \\ \sqrt{\frac{2}{3}} \end{pmatrix} \bigotimes \begin{pmatrix} 1 & 1 & 1 \\ -\sin\left(\theta+\frac{\pi}{6}\right) & \cos(\theta) & \sin\left(\theta-\frac{\pi}{6}\right) \\ -\cos\left(\theta+\frac{\pi}{6}\right) & -\sin(\theta) & \cos\left(\theta-\frac{\pi}{6}\right) \end{pmatrix} \tag{A.12}$$

To facilitate the quantization, we next factor the second and third rows such that the largest element of each row equals exactly 1. As previously mentioned, the largest element has the opposite sign to the other two elements in a given row. So, the steps required to

perform the factorization are as follows:

- Find the sign of each of the elements.
- Determine which of those elements is of the opposite sign.
- Factor each row by that largest element.

The sign of the elements is most easily seen by looking at the rotation as phase shifted sine functions.

$$\mathbf{rR}(\theta) = \begin{pmatrix} 1 & 1 & 1 \\ \sin\left(\theta - \frac{5\pi}{6}\right) & \sin\left(\theta + \frac{3\pi}{6}\right) & \sin\left(\theta - \frac{\pi}{6}\right) \\ \sin\left(\theta - \frac{2\pi}{6}\right) & \sin\left(\theta - \frac{6\pi}{6}\right) & \sin\left(\theta + \frac{2\pi}{6}\right) \end{pmatrix} \tag{A.13}$$

So the signs look like

$$\begin{pmatrix} 1 & 1 & 1 \\ \begin{cases} 1 & \frac{5\pi}{6} \le \theta < \frac{11\pi}{6} \\ -1 & -\frac{\pi}{6} \le \theta < \frac{5\pi}{6} \end{cases} & \begin{cases} 1 & -\frac{\pi}{2} \le \theta < \frac{\pi}{2} \\ -1 & \frac{\pi}{2} \le \theta < \frac{3\pi}{2} \end{cases} & \begin{cases} 1 & \frac{\pi}{6} \le \theta < \frac{7\pi}{6} \\ -1 & -\frac{5\pi}{6} \le \theta < \frac{\pi}{6} \end{cases} \\ \begin{cases} 1 & \frac{\pi}{3} \le \theta < \frac{4\pi}{3} \\ -1 & -\frac{2\pi}{3} \le \theta < \frac{\pi}{3} \end{cases} & \begin{cases} 1 & -\pi \le \theta < 0 \\ -1 & 0 \le \theta < \pi \end{cases} & \begin{cases} 1 & -\frac{\pi}{3} \le \theta < \frac{2\pi}{3} \\ -1 & \frac{2\pi}{3} \le \theta < \frac{5\pi}{3} \end{cases} \end{pmatrix} \tag{A.14}$$

Each row is a series of three sin waves each $\frac{2\pi}{3}$ out of phase with each other. The rows are $\frac{\pi}{2}$ out of phase with each other. The largest element changes every $\frac{\pi}{6}$ radians repeating every π radians.

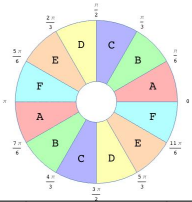
	Scale : $fS[\theta]$	Factored Rotation : $fR[\theta]$
		
A	$\begin{pmatrix} 1 \\ \cos(\theta) \\ \cos\left(\frac{\pi}{6}-\theta\right) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ -\sec(\theta)\sin\left(\theta+\frac{\pi}{6}\right) & 1 & \sec(\theta)\sin\left(\theta-\frac{\pi}{6}\right) \\ -\cos\left(\theta+\frac{\pi}{6}\right)\sec\left(\frac{\pi}{6}-\theta\right) & -\sec\left(\frac{\pi}{6}-\theta\right)\sin(\theta) & 1 \end{pmatrix}$
B	$\begin{pmatrix} 1 \\ -\sin\left(\theta+\frac{\pi}{6}\right) \\ \cos\left(\frac{\pi}{6}-\theta\right) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -\cos(\theta)\csc\left(\theta+\frac{\pi}{6}\right) & \csc\left(\theta+\frac{\pi}{6}\right)\sin\left(\frac{\pi}{6}-\theta\right) \\ -\cos\left(\theta+\frac{\pi}{6}\right)\sec\left(\frac{\pi}{6}-\theta\right) & -\sec\left(\frac{\pi}{6}-\theta\right)\sin(\theta) & 1 \end{pmatrix}$
C	$\begin{pmatrix} 1 \\ -\sin\left(\theta+\frac{\pi}{6}\right) \\ -\sin(\theta) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -\cos(\theta)\csc\left(\theta+\frac{\pi}{6}\right) & \csc\left(\theta+\frac{\pi}{6}\right)\sin\left(\frac{\pi}{6}-\theta\right) \\ \cos\left(\theta+\frac{\pi}{6}\right)\csc(\theta) & 1 & -\cos\left(\frac{\pi}{6}-\theta\right)\csc(\theta) \end{pmatrix}$
D	$\begin{pmatrix} 1 \\ \sin\left(\theta-\frac{\pi}{6}\right) \\ -\sin(\theta) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ \csc\left(\frac{\pi}{6}-\theta\right)\sin\left(\theta+\frac{\pi}{6}\right) & -\cos(\theta)\csc\left(\frac{\pi}{6}-\theta\right) & 1 \\ \cos\left(\theta+\frac{\pi}{6}\right)\csc(\theta) & 1 & -\cos\left(\frac{\pi}{6}-\theta\right)\csc(\theta) \end{pmatrix}$
E	$\begin{pmatrix} 1 \\ \sin\left(\theta-\frac{\pi}{6}\right) \\ -\cos\left(\theta+\frac{\pi}{6}\right) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ \csc\left(\frac{\pi}{6}-\theta\right)\sin\left(\theta+\frac{\pi}{6}\right) & -\cos(\theta)\csc\left(\frac{\pi}{6}-\theta\right) & 1 \\ 1 & \sec\left(\theta+\frac{\pi}{6}\right)\sin(\theta) & -\cos\left(\frac{\pi}{6}-\theta\right)\sec\left(\theta+\frac{\pi}{6}\right) \end{pmatrix}$
F	$\begin{pmatrix} 1 \\ \cos(\theta) \\ -\cos\left(\theta+\frac{\pi}{6}\right) \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ -\sec(\theta)\sin\left(\theta+\frac{\pi}{6}\right) & 1 & \sec(\theta)\sin\left(\theta-\frac{\pi}{6}\right) \\ 1 & \sec\left(\theta+\frac{\pi}{6}\right)\sin(\theta) & -\cos\left(\frac{\pi}{6}-\theta\right)\sec\left(\theta+\frac{\pi}{6}\right) \end{pmatrix}$

Table A.1 Table of factored rotation matrices.

By defining θ as $\theta = \Theta + \theta_1$ — where $\theta_1 = \theta \mod \frac{\pi}{6}$, and Θ is the starting value for the region in which θ lies — and substituting in the function $\mathbf{fRe}(\phi)$:

$$\mathbf{fRe}(\phi) = \frac{-1}{2}\left(1+\sqrt{3}\tan(\phi)\right) \tag{A.15}$$

we can now rewrite the factored rotation as shown in Table A.2.

We can now relate the functions to each other, since they are now essentially the same function, and θ_1 has the same domain; the values they take have the same range, despite being in different positions in the matrix. This allows us to write the piecewise function **fR** independently as a function **fRO**, which re-orders the elements of the matrix in region **A** appropriately for the other regions.

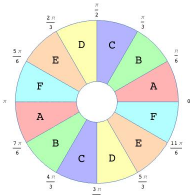
Θ 	Scale : $fS[\theta]$ $\vec{fS}s(\theta) \otimes \mathrm{fSe}(\theta)$	Factored Rotation $fR[\theta]$ where $\theta_1 = \theta \mod \frac{\pi}{6}$
A : 0	$\begin{pmatrix}1\\1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos(\theta_1)\\\cos\left(\frac{\pi}{6}-\theta_1\right)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ \mathrm{fRe}(\theta_1) & 1 & \mathrm{fRe}(-\theta_1)\\ \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right) & \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right) & 1\end{pmatrix}$
B : $\frac{\pi}{6}$	$\begin{pmatrix}1\\-1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos\left(\frac{\pi}{6}-\theta_1\right)\\\cos(\theta_1)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right) & \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right)\\ \mathrm{fRe}(-\theta_1) & \mathrm{fRe}(\theta_1) & 1\end{pmatrix}$
C : $\frac{\pi}{3}$	$\begin{pmatrix}1\\-1\\-1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos(\theta_1)\\\cos\left(\frac{\pi}{6}-\theta_1\right)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & \mathrm{fRe}(-\theta_1) & \mathrm{fRe}(\theta_1)\\ \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right) & 1 & \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right)\end{pmatrix}$
D : $\frac{\pi}{2}$	$\begin{pmatrix}1\\1\\-1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos\left(\frac{\pi}{6}-\theta_1\right)\\\cos(\theta_1)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right) & \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right) & 1\\ \mathrm{fRe}(\theta_1) & 1 & \mathrm{fRe}(-\theta_1)\end{pmatrix}$
E : $\frac{2\pi}{3}$	$\begin{pmatrix}1\\1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos(\theta_1)\\\cos\left(\frac{\pi}{6}-\theta_1\right)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ \mathrm{fRe}(-\theta_1) & \mathrm{fRe}(\theta_1) & 1\\ 1 & \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right) & \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right)\end{pmatrix}$
F : $\frac{5\pi}{6}$	$\begin{pmatrix}1\\-1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\\cos\left(\frac{\pi}{6}-\theta_1\right)\\\cos(\theta_1)\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right) & 1 & \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right)\\ 1 & \mathrm{fRe}(-\theta_1) & \mathrm{fRe}(\theta_1)\end{pmatrix}$

Table A.2 Simplified table of factored rotation matrices.

$$\mathbf{fR}(\theta) = \mathbf{fRO}[\mathbf{fRm}, \theta] \qquad \mathbf{fRm} = \begin{pmatrix}1 & 1 & 1\\ \mathrm{fRe}(\theta_1) & 1 & \mathrm{fRe}(-\theta_1)\\ \mathrm{fRe}\left(\frac{\pi}{6}-\theta_1\right) & \mathrm{fRe}\left(\theta_1-\frac{\pi}{6}\right) & 1\end{pmatrix} \tag{A.16}$$

The scaling factor $\vec{fS}(\theta_1)$ can also be further simplified by separating the sign and combining the functional parts.

$$\mathrm{fSe}(\theta) = \begin{cases} \begin{pmatrix}1\\\cos\left(\frac{\pi}{6}-\theta_1\right)\\\cos(\theta_1)\end{pmatrix} & \frac{\pi}{6} \le (\theta \bmod \pi) < \frac{\pi}{3} \vee \\ & \frac{\pi}{2} \le (\theta \bmod \pi) < \frac{2\pi}{3} \vee \\ & \frac{5\pi}{6} \le (\theta \bmod \pi) < \pi \\ \begin{pmatrix}1\\\cos(\theta_1)\\\cos\left(\frac{\pi}{6}-\theta_1\right)\end{pmatrix} & 0 \le (\theta \bmod \pi) < \frac{\pi}{6} \vee \\ & \frac{\pi}{3} \le (\theta \bmod \pi) < \frac{\pi}{2} \vee \\ & \frac{2\pi}{3} \le (\theta \bmod \pi) < \frac{5\pi}{6} \end{cases} \tag{A.17}$$

This can be simplified into

$$\mathrm{fSe}(\theta) = \begin{pmatrix} 1 \\ \cos\left(\frac{\pi}{6} - \widetilde{\vartheta}\right) \\ \cos\left(\frac{\pi}{6} - \widetilde{\theta}\right) \end{pmatrix} \quad \text{where} \quad \begin{matrix} \widetilde{\vartheta} = \left(\theta - \frac{\pi}{6}\right) \bmod \frac{\pi}{3} \\ \widetilde{\theta} = \theta \bmod \frac{\pi}{3} \end{matrix} \tag{A.18}$$

then $\vec{fS}(\theta) = \mathrm{fSe}(\theta) \otimes \vec{fSs}(\theta)$, where $\vec{fSs}$ is a piecewise function containing the signs.

The scaling factors combine with the normalisation scaling factor A.7, found in the previous section, in a simple way

$$\vec{S} = \vec{L^{-1}}(\theta) \otimes \vec{rS} \otimes \mathrm{fSe}(\theta) = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \tag{A.19}$$

The fully factorized rotation matrix can now be written as

$$\mathbf{R}(\theta) = \vec{L}(\theta) \otimes \vec{L^{-1}}(\theta) \otimes \vec{rS} \otimes \mathrm{fSe}(\theta) \otimes \vec{fSs}(\theta) \otimes \mathbf{fRO}[\mathbf{fR}\mathrm{m}, \theta]$$

and so

$$\mathbf{R}(\theta) = \vec{L}(\theta) \otimes \vec{S} \otimes \vec{fSs}(\theta) \otimes \mathbf{fRO}[\mathbf{fR}\mathrm{m}, \theta] \tag{A.20}$$

where (A.21)

$$\mathbf{L}(\theta) = \begin{pmatrix} \sqrt{3} \\ \sqrt{\frac{2}{3}}\sin\left(\widetilde{\vartheta}\right) + \sqrt{2}\cos\left(\widetilde{\vartheta}\right) \\ \sqrt{\frac{2}{3}}\sin\left(\widetilde{\theta}\right) + \sqrt{2}\cos\left(\widetilde{\theta}\right) \end{pmatrix} \quad \text{with} \quad \begin{matrix} \widetilde{\vartheta} = \left(\theta - \frac{\pi}{6}\right) \bmod \frac{\pi}{3} \\ \widetilde{\theta} = \theta \bmod \frac{\pi}{3} \end{matrix} \tag{A.22}$$

$$\mathbf{fR}\mathrm{m} = \begin{pmatrix} 1 & 1 & 1 \\ \mathrm{fRe}(\theta_1) & 1 & \mathrm{fRe}(-\theta_1) \\ \mathrm{fRe}\left(\frac{\pi}{6} - \theta_1\right) & \mathrm{fRe}\left(\theta_1 - \frac{\pi}{6}\right) & 1 \end{pmatrix} \quad \text{with} \quad \begin{matrix} \mathbf{fR}\mathrm{e}(\phi) = \frac{-1}{2}\left(1 + \sqrt{3}\tan(\phi)\right) \\ \theta_1 = \theta \mod \frac{\pi}{6} \end{matrix} \tag{A.23}$$

$$\vec{S} = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \qquad f\vec{S}s(\theta) = \begin{cases} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} & 0 \leq (\theta \bmod \frac{2\pi}{3}) < \frac{\pi}{6} \\ \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} & \frac{\pi}{6} \leq (\theta \bmod \frac{2\pi}{3}) < \frac{\pi}{3} \\ \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} & \frac{\pi}{3} \leq (\theta \bmod \frac{2\pi}{3}) < \frac{\pi}{2} \\ \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} & \frac{\pi}{2} \leq (\theta \bmod \frac{2\pi}{3}) < \frac{2\pi}{3} \end{cases} \tag{A.24}$$

$$\mathbf{fRO}\left[\begin{pmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \\ C_1 & C_2 & C_3 \end{pmatrix}, \theta\right] = \begin{cases} \begin{pmatrix} A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \\ C_1 & C_2 & C_3 \end{pmatrix} & 0 \leq (\theta \bmod \pi) < \frac{\pi}{6} \\ \begin{pmatrix} A_3 & A_1 & A_2 \\ C_3 & C_1 & C_2 \\ B_3 & B_1 & B_2 \end{pmatrix} & \frac{\pi}{6} \leq (\theta \bmod \pi) < \frac{\pi}{3} \\ \begin{pmatrix} A_2 & A_3 & A_1 \\ B_2 & B_3 & B_1 \\ C_2 & C_3 & C_1 \end{pmatrix} & \frac{\pi}{3} \leq (\theta \bmod \pi) < \frac{\pi}{2} \\ \begin{pmatrix} A_1 & A_2 & A_3 \\ C_1 & C_2 & C_3 \\ B_1 & B_2 & B_3 \end{pmatrix} & \frac{\pi}{2} \leq (\theta \bmod \pi) < \frac{2\pi}{3} \\ \begin{pmatrix} A_3 & A_1 & A_2 \\ B_3 & B_1 & B_2 \\ C_3 & C_1 & C_2 \end{pmatrix} & \frac{2\pi}{3} \leq (\theta \bmod \pi) < \frac{5\pi}{6} \\ \begin{pmatrix} A_2 & A_3 & A_1 \\ C_2 & C_3 & C_1 \\ B_2 & B_3 & B_1 \end{pmatrix} & \frac{5\pi}{6} \leq (\theta \bmod \pi) < \pi \end{cases} \tag{A.25}$$

This achieves our goal of factoring the rotation such that the largest element of each matrix row is one.

Appendix B

Quantizing the Rotation

The factored rotation **fR** A.20 contains 4 non-integer elements lying between -1 and 0, which we wish to quantize.

$$\mathbf{R}(\theta) = \vec{L}(\theta) \otimes \vec{S} \otimes f\vec{S}s(\theta) \otimes \mathbf{fR} \qquad \textbf{where} \qquad \mathbf{fR} = \mathbf{fR}\mathrm{O}\left[\mathbf{fR}\mathrm{m}, \theta\right] \tag{B.1}$$

This can be done simply by multiplying it by 2^{n-2}, where n is the bit depth of the integer type, and rounding the result:

$$\mathbf{qR}(\theta, n) = \mathbf{Round}\left[\begin{pmatrix} 1 \\ 2^{n-2} \\ 2^{n-2} \end{pmatrix} \bigotimes \mathbf{fR}(\theta)\right] \qquad \text{and} \quad \vec{qS}(n) = \begin{pmatrix} 1 \\ 2^{2-n} \\ 2^{2-n} \end{pmatrix} \tag{B.2}$$

Since every row in **fR** always has an element equal to 1, we will exploit the full range $-2^{n-2} \leq \mathbf{fR} \leq 2^{n-2}$ of the integer type.

$$\mathbf{fR}(\theta) = \vec{qS}(n) \otimes (\mathbf{qR}(\theta, n) + \delta\mathbf{qR}(\theta, n)) \tag{B.3}$$

$$= \vec{qS}(n) \otimes \mathbf{qR}(\theta, n) + \delta\mathbf{fR}(\theta, n) \tag{B.4}$$

Thus far, we have assumed that we can achieve a bit depth of two less than the bit depth of the source type for the representation of the rotation matrix. We are now in a position to assess whether the representation introduces any errors.

Given that the source type with a bit depth of n, we can work out any errors introduced by

finding the difference between using the quantized rotation **qR** and the unquantized rotation **fR**. All the factoring of the rotation is mathematically exact up to and including **fR**, so any errors that are introduced will be during the rounding.

We are interested in the error introduced by the quantization, so we introduce the perturbation $\delta\mathbf{fR}(\theta)$ associated with the transform $\mathbf{fR}(\theta)$ produced by the rounding. Defining

$$\mathbf{fR}\mathrm{m}(\theta) = \vec{qS}(n) \otimes (\mathbf{qR}\mathrm{m}(\theta,n) + \delta\mathbf{qR}\mathrm{m}(\theta,n))$$

because $\vec{qS}$ has equal elements in the second and third rows

$$\mathbf{fRO}\left[\vec{qS}(n) \otimes \mathbf{A}\right] = \vec{qS}(n) \otimes \mathbf{fRO}[\mathbf{A}]$$

then

$$\begin{aligned}
\mathbf{fR}(\theta) &= \mathbf{fRO}[\mathbf{fR}\mathrm{m}(\theta)] \\
&= \mathbf{fRO}\left[\vec{qS}(n) \otimes (\mathbf{qR}\mathrm{m}(\theta,n) + \delta\mathbf{qR}\mathrm{m}(\theta,n))\right] \\
&= \vec{qS}(n) \otimes \mathbf{fRO}[\mathbf{qR}\mathrm{m}(\theta,n) + \delta\mathbf{qR}\mathrm{m}(\theta,n)] \\
&= \vec{qS}(n) \otimes \mathbf{fRO}[\mathbf{qR}\mathrm{m}(\theta,n)] + \vec{qS}(n) \otimes \mathbf{fRO}[\delta\mathbf{qR}\mathrm{m}(\theta,n)]
\end{aligned}$$

The perturbation depends entirely on $\delta\mathbf{qR}\mathrm{m}(\theta,n)$

$$\begin{aligned}
\delta\mathbf{qR}\mathrm{e}(\phi) &= \mathrm{Round}\left[2^{n-2}\mathbf{fR}\mathrm{e}(\phi)\right] - 2^{n-2}\mathbf{fR}\mathrm{e}(\phi) \\
&= \mathrm{Round}\left[2^{n-2}\frac{-1}{2}\left(1+\sqrt{3}\tan(\phi)\right)\right] - 2^{n-2}\frac{-1}{2}\left(1+\sqrt{3}\tan(\phi)\right) \\
&= \mathrm{Round}\left[2^{n-3}\right] + \mathrm{Round}\left[2^{n-3}\sqrt{3}\tan(\phi)\right] - 2^{n-3} - 2^{n-3}\sqrt{3}\tan(\phi) \\
&= \mathrm{Round}\left[2^{n-3}\sqrt{3}\tan(\phi)\right] - 2^{n-3}\sqrt{3}\tan(\phi) \\
\delta\mathbf{fR}\mathrm{e}(\phi) &= 2^{2-n}\delta\mathbf{qR}\mathrm{e}(\phi)
\end{aligned}$$

The perturbation $\delta\mathbf{qR}(\theta)$ is a rearrangement of $\delta\mathbf{qR}\mathrm{m} = \delta\mathbf{qR}(\theta_1)$ where $\theta_1 = \theta \bmod \frac{\pi}{6}$

$$\begin{aligned}\delta\mathbf{qR}(\theta) &= \mathbf{fRO}[\delta\mathbf{qR}\mathrm{m}, \theta] \\ &= \mathbf{fRO}\left[\begin{pmatrix} 0 & 0 & 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) & 0 & \delta\mathbf{qR}\mathrm{e}(-\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) & \delta\mathbf{qR}\mathrm{e}\left(\theta_1-\frac{\pi}{6}\right) & 0 \end{pmatrix}, \theta\right]\end{aligned}$$

The perturbation introduced to each channel is further simplified by recognizing that $\mathbf{fR}\mathrm{e}(-\phi) = -1 - \mathbf{fR}\mathrm{e}(\phi)$, which allows us to find $\delta\mathbf{qR}\mathrm{e}(-\phi) = -\delta\mathbf{qR}\mathrm{e}(\phi)$ i.e. that the sum of the two perturbations in each row equals zero. we can analyse the quantization error in terms of just one function $\delta\mathbf{qR}\mathrm{e}$ and a matrix ordering function $\delta\mathbf{qR}(\theta) = \mathbf{fRO}[\delta\mathbf{qR}(\theta_1), \theta]$.

$$\begin{aligned}\delta\mathbf{qR}(\theta) &= \widehat{\mathbf{fRO}}\left[\begin{pmatrix} 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{pmatrix}, \theta\right] \\ &= \widehat{\delta\mathbf{qEO}}\left[\begin{pmatrix} 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{pmatrix}\right] \otimes \delta\mathbf{qS}(\theta)\end{aligned}$$

where

$$\widehat{\delta\mathbf{qEO}}\left[\begin{pmatrix} 0 \\ a \\ b \end{pmatrix}, \theta\right] = \begin{cases} \begin{pmatrix} 0 \\ a \\ b \end{pmatrix} & 0 \le (\theta \bmod \frac{\pi}{3}) < \frac{\pi}{6} \\ \begin{pmatrix} 0 \\ b \\ a \end{pmatrix} & \frac{\pi}{6} \le (\theta \bmod \frac{\pi}{3}) < \frac{\pi}{3} \end{cases} \tag{B.5}$$

$$\delta\mathbf{qS}(\theta) = \begin{cases} \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{pmatrix} & 0 \le (\theta \bmod \pi) < \frac{\pi}{6} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{pmatrix} & \frac{\pi}{6} \le (\theta \bmod \pi) < \frac{\pi}{3} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 1 \end{pmatrix} & \frac{\pi}{3} \le (\theta \bmod \pi) < \frac{\pi}{2} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix} & \frac{\pi}{2} \le (\theta \bmod \pi) < \frac{2\pi}{3} \\ \begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{pmatrix} & \frac{2\pi}{3} \le (\theta \bmod \pi) < \frac{5\pi}{6} \\ \begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix} & \frac{5\pi}{6} \le (\theta \bmod \pi) < \pi \end{cases} \tag{B.6}$$

We can now express the perturbation to the rotation and to the channel elements themselves. The structure of $\widehat{\mathbf{fRO}}$ means that the perturbation to each channel is the product

of either $\delta\mathbf{qR}\mathrm{e}(\theta_1)$ or $\delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right)$ with the difference between two of the pixel values. Given that we can assume that the pixel values are positive, being unsigned integers, the extreme values of the perturbation are when one of the two pixel values is zero.

$$\overline{\mathbf{R}}(\theta) = \widetilde{\overline{\mathbf{R}}}(\theta) + \overline{\delta\mathbf{R}}(\theta) \tag{B.7}$$

where (B.8)

$$\widetilde{\overline{\mathbf{R}}}(\theta) = \vec{S} \otimes f\vec{S}s(\theta) \otimes \vec{qS}(n) \otimes \mathbf{fRO}\left[\mathbf{qR}\mathrm{m}(\theta, n), \theta\right] \tag{B.9}$$

$$\overline{\delta\mathbf{R}}(\theta) = \vec{S} \otimes f\vec{S}s(\theta) \otimes \vec{qS}(n) \otimes \widehat{\delta\mathbf{qEO}} \begin{bmatrix} 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{bmatrix} \otimes \delta\mathbf{qS}(\theta) \tag{B.10}$$

the perturbation to the rotated channel elements $\vec{\delta w}$ is found for an input set of pixel values $\vec{v} = 2^n\vec{\overline{v}}$ where $0 \leq \vec{\overline{v}} \leq 1$

$$\vec{w} + \vec{\delta w} = \widetilde{\overline{\mathbf{R}}}(\theta) \cdot \vec{v} + \overline{\delta\mathbf{R}}(\theta) \cdot \vec{v}$$
$$\vec{\delta w} = 2^n\, \overline{\delta\mathbf{R}}(\theta) \cdot \vec{\overline{v}}$$

Expanding $\overline{\delta\mathbf{R}}(\theta)$ using B.10.

$$\begin{aligned}
\vec{\delta w} &= 2^n\, \vec{S} \otimes \vec{qS}(n) \otimes f\vec{S}s(\theta) \otimes \widehat{\delta\mathbf{qEO}} \begin{bmatrix} 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{bmatrix} \otimes \delta\mathbf{qS}(\theta) \cdot \vec{\overline{v}} \\
&= \begin{pmatrix} \frac{2^n}{3} \\ 2 \\ 2 \end{pmatrix} \otimes \widehat{\delta\mathbf{qEO}} \begin{bmatrix} 0 \\ \delta\mathbf{qR}\mathrm{e}(\theta_1) \\ \delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{bmatrix} \otimes f\vec{S}s(\theta) \otimes \delta\mathbf{qS}(\theta) \cdot \vec{\overline{v}} \\
&= \widehat{\delta\mathbf{qEO}} \begin{bmatrix} 0 \\ 2\delta\mathbf{qR}\mathrm{e}(\theta_1) \\ 2\delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) \end{bmatrix} \otimes f\vec{S}s(\theta) \otimes \delta\mathbf{qS}(\theta) \cdot \vec{\overline{v}}
\end{aligned}$$

So, to minimize the perturbation, we need to simultaneously minimize $\pm 2\delta\mathbf{qR}\mathrm{e}(\theta_1)$ and $\pm 2\delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right)$

The maxima for each function are where $\pm 2\delta\mathbf{qR}\mathrm{e}(\theta_1) = \pm 1$ and $\pm 2\delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) = \pm 1$, and the minima are where $\pm 2\delta\mathbf{qR}\mathrm{e}(\theta_1) = 0$ and $\pm 2\delta\mathbf{qR}\mathrm{e}\left(\frac{\pi}{6}-\theta_1\right) = 0$. To find the corresponding values of θ_1, all that is needed is an inverse function for $\delta\mathbf{qR}\mathrm{e}$. In essence $\delta\mathbf{qR}\mathrm{e}(\phi)$ discretizes $\sqrt{3}\tan(\phi)$ into steps of 2^{3-n}. The minima can be found by taking

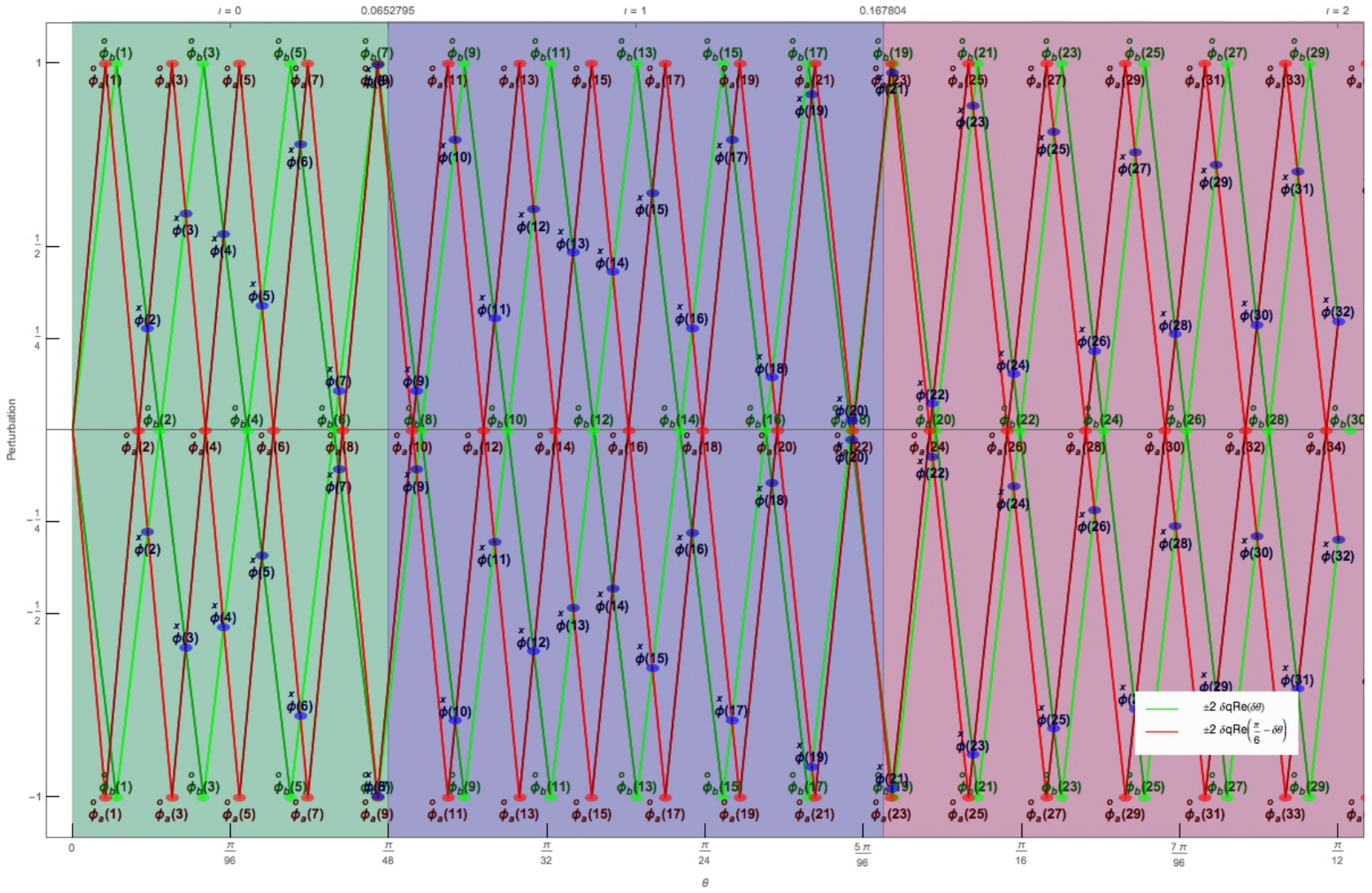


Fig. B.1 Perturbation to channels a and b against angle θ_1
For n = 8 with ι regions shaded and extrema labelled

the inverse of $\sqrt{3}\tan(\phi)$, given that $0 \leq \sqrt{3}\tan(\phi) \leq 1$ for $0 \leq \phi \leq \frac{\pi}{6}$ we can say that Round $\left[2^{n-3}\sqrt{3}\tan(\phi)\right] \in \{0,1,2,\cdots 2^{n-3}\}$

$$\delta\mathbf{qRe}(\phi) = 0 \qquad \delta\mathbf{qRe}\left(\frac{\pi}{6}-\phi\right) = 0 \quad \text{when}$$

$$2^{n-3}\sqrt{3}\tan(\phi) = m \qquad 2^{n-3}\sqrt{3}\tan(\frac{\pi}{6}-\phi) = m \quad \text{where} \quad m = 0,1,2\cdots 2^{n-3}$$

$$\check{\phi}_b = \arctan\left(\frac{m2^{3-n}}{\sqrt{3}}\right) \qquad \check{\phi}_a = \arctan\left(\frac{2\sqrt{3}m}{2^n-2m}\right)$$

The maxima can be found by recognizing that the maximum rounding error is a half and evaluating at these points

$$\delta\mathbf{qRe}\left(\hat{\phi}_b\right) = \pm 1 \qquad \delta\mathbf{qRe}\left(\frac{\pi}{6}-\hat{\phi}_a\right) = \pm 1 \quad \text{when}$$

$$\hat{\phi}_b = \arctan\left(\frac{2^{3-n}}{\sqrt{3}}\left(m+\frac{1}{2}\right)\right) \qquad \hat{\phi}_a = \arctan\left(\frac{2\sqrt{3}\left(m+\frac{1}{2}\right)}{2^n-2\left(m+\frac{1}{2}\right)}\right)$$

$$\hat{\phi}_b = \arctan\left(\frac{(2m+1)2^{2-n}}{\sqrt{3}}\right) \qquad \hat{\phi}_a = \arctan\left(\frac{\sqrt{3}(2m+1)}{2^n-2m-1}\right)$$

$$\text{with} \quad m = 0,1,2,3\cdots 2^{n-3}-1$$

We are interested in a compromise between the perturbations to each channel which is where the perturbations intersect. It would be useful to be able to assign a relative importance α and β to each channel a and b respectively.

$$\beta\ \delta\mathbf{qRe}(\phi) = \alpha\ \delta\mathbf{qRe}\left(\frac{\pi}{6}-\phi\right) \quad \textbf{when}$$

$$\beta\text{Round}\left[2^{n-3}\sqrt{3}\tan(\phi)\right] - \alpha\text{Round}\left[2^{n-3}\sqrt{3}\tan(\frac{\pi}{6}-\phi)\right]$$

$$= 2^{n-3}\left(\sqrt{3}\alpha\tan(\theta) + 2\beta\sin(\theta)\sec\left(\frac{\pi}{6}-\theta\right)\right)$$

The values α and β can only move the point of intersection within the bounds of the extrema. Therefore, any solution for the point of intersection will enable the bounding extrema to be identified.

We solve first for the case where $\alpha = 1$ and $\beta = 1$. Recognizing that the rounded part

can only take certain values

$$i = \text{Round}\left[2^{n-3}\sqrt{3}\tan(\phi)\right] - \text{Round}\left[2^{n-3}\sqrt{3}\tan(\frac{\pi}{6}-\phi)\right] \quad \text{where} \quad i = 0,1,2\cdots 2^{n-2} \tag{B.11}$$

then

$$\overset{\times}{\phi}(i) = \arctan\left(\frac{\sqrt{i^2 2^{6-2n} - i2^{4-n} + 49} + i2^{3-n} - 7}{2\sqrt{3}}\right) \quad \text{where} \quad i = 0,1,2\cdots 2^{n-2}$$

Each point of intersection is defined and bounded by 4 extrema. Comparing with the functional form of the extrema allows these 4 points to be identified.

$$\mathring{\phi}_a(l) = \arctan\left(\frac{l\sqrt{3}}{2^n - l}\right) \quad \mathring{\phi}_b(l) = \arctan\left(\frac{l\,2^{2-n}}{\sqrt{3}}\right) \tag{B.12}$$

$$\check{\phi}_a = \mathring{\phi}_a(2m) \qquad \check{\phi}_b = \mathring{\phi}_b(2m) \qquad \text{with} \quad m = 0,1,2,3\cdots 2^{n-3} \tag{B.13}$$

$$\hat{\phi}_a = \mathring{\phi}_a(2m+1) \qquad \hat{\phi}_b = \mathring{\phi}_b(2m+1) \qquad \text{with} \quad m = 0,1,2,3\cdots 2^{n-3}-1 \tag{B.14}$$

Separating the points of intersection into even and odd terms facilitates the identification.

	Channel a	Channel b	i
Minima	$\check{\phi}_a(\frac{i}{2}) = \mathring{\phi}_a(i)$	$\check{\phi}_b(\frac{i}{2}) = \mathring{\phi}_b(i)$	$i \in \{0,2,4,6\cdots 2^{n-2}\}$
Maxima	$\hat{\phi}_a(\frac{i-1}{2}) = \mathring{\phi}_a(i)$	$\hat{\phi}_b(\frac{i-1}{2}) = \mathring{\phi}_b(i)$	$i \in \{1,3,5,7\cdots 2^{n-2}-1\}$

The following ranges are guaranteed to be true for even or odd values of i, with very specific values of p and q.

$$\begin{array}{ccccc} \mathring{\phi}_b(p-1) & < \overset{\times}{\phi}(i) < & \mathring{\phi}_a(q+1) & \bigvee & \mathring{\phi}_a(q+1) < \overset{\times}{\phi}(i+1) < \mathring{\phi}_b(p+1) \\ \mathring{\phi}_a(q) & < & \mathring{\phi}_b(p) & & \mathring{\phi}_b(p) < \quad < \mathring{\phi}_a(q+2) \end{array} \qquad \begin{array}{rl} p & \in \{2\mathbb{N}\} \\ q & \in \{2\mathbb{N}\} \\ 2 & \leq i \leq 2^{n-2}-1 \end{array}$$

This slightly glib definition illustrates the pattern for the extrema bounding the points of intersection. At the start of a $\frac{\Pi}{6}$ region, i.e. for low values of i, $p = i$ and $q = i$. The maxima swap orderings between certain values of i within a $\frac{\Pi}{6}$ region. This is evident in the inescapable indices of $i-1$ and $i+1$, regardless of the starting values for i for either the

maxima or minima. So, whilst it is true that $\mathring{\phi}_a(i) > \mathring{\phi}_b(i)$, it is not always true that $\mathring{\phi}_a(i+1) > \mathring{\phi}_b(i-1)$. This leads to the question of what happens when this boundary $\mathring{\phi}_a(i+1) = \mathring{\phi}_b(i-1)$ is crossed. Firstly, the even-odd ordering (i.e. the a-b channel ordering) is flipped; secondly, the indices are shifted. A second transition restores the a-b ordering and shifts the indices again. As the index i passes the half way point $\bar{\mathbf{i}} = 2^{n-3}$, the transitions occur in the opposite direction, restoring the indices and ordering. Transitions occur where i satisfies $\mathring{\phi}_a(i+\imath) = \mathring{\phi}_b(i-\imath)$, with $\imath \in \{1,2,3\cdots\left(7-4\sqrt{3}\right)2^{n-3}\}$.

$$\bar{\mathbf{i}} = 2^{n-3} \qquad \Gamma(\imath,n) = \sqrt{\imath^2 - 14\imath\bar{\mathbf{i}} + \bar{\mathbf{i}}^2}$$

The region $\imath$ for a given value of i is the highest value of $\imath$ for which the following condition remains true:

$$\bar{\mathbf{i}} - \Gamma(\imath,n) \leq i \leq \Gamma(\imath,n) + \bar{\mathbf{i}}$$

For a given value of θ and n, we can find the intersection index i, and from this we can find $\imath$:

$$i(\theta_1,n) = \frac{\bar{\mathbf{i}}\tan(\theta_1)\left(\sqrt{3}\tan(\theta_1)+7\right)}{\tan(\theta_1)+\sqrt{3}} \quad \text{where} \quad \theta_1 = \theta \bmod \frac{\pi}{6} \tag{B.15}$$

$$\imath(i,n) = \left\lfloor 7\,\bar{\mathbf{i}} - \sqrt{i^2 - 2i\,\bar{\mathbf{i}} + 49\,\bar{\mathbf{i}}^2} \right\rfloor \tag{B.16}$$

The sequence of a-b channel extrema depends on both the index i and the region $\imath$ in the following way:

$$\begin{array}{ccccc} \mathring{\phi}_b(i-\imath-1) & < & \overset{\times}{\phi}(i) & < & \mathring{\phi}_a(i+\imath+1) \\ \mathring{\phi}_a(i+\imath) & < & & < & \mathring{\phi}_b(i-\imath) \end{array} \quad \text{when} \quad \begin{array}{l} i \in \{2\mathbb{N}+1\} \ \wedge \ \imath \in \{2\mathbb{N}+1\} \vee \\ i \in \{2\mathbb{N}\} \ \wedge \ \imath \in \{2\mathbb{N}\} \end{array} \tag{B.17}$$

$$\begin{array}{ccccc} \mathring{\phi}_a(i+\imath) & < & \overset{\times}{\phi}(i) & < & \mathring{\phi}_b(i-\imath) \\ \mathring{\phi}_b(i-\imath-1) & < & & < & \mathring{\phi}_a(i+\imath+1) \end{array} \quad \text{when} \quad \begin{array}{l} (i \in \{2\mathbb{N}+1\} \ \wedge \ \imath \in \{2\mathbb{N}\}) \vee \\ (i \in \{2\mathbb{N}\} \ \wedge \ \imath \in \{2\mathbb{N}+1\}) \end{array} \tag{B.18}$$

The other two ranges can swap direction around the central value $\overline{m}$ in a region governed by Γ which is the distance from $\overline{m}$ at which the order of the extrema switch between channels.

$$\overline{m} = \frac{1}{2}\left(2^{n-3}-1\right) \qquad \Gamma(n) = \frac{1}{2}\sqrt{2^{2(n-3)} - 14\,2^{n-3} + 1}$$

$$\lessgtr_m = \begin{cases} < & m < \bar{m} - \Gamma \vee m > \bar{m} + \Gamma \\ > & \bar{m} - \Gamma \leq m \leq \bar{m} + \Gamma \end{cases}$$

This allows the bounds for the points of intersection to be concisely written

$$\hat{\phi}_b(m-1) \underset{(m-\frac{1}{2})}{\lessgtr} \overset{\times}{\phi}(2m) \underset{(m-\frac{1}{2})}{\lessgtr} \hat{\phi}_a(m)$$

$$\check{\phi}_b(m) \underset{m}{\lessgtr} \overset{\times}{\phi}(2m+1) \underset{m}{\lessgtr} \check{\phi}_a(m+1)$$

or

$$\begin{array}{ccccc} \hat{\phi}_b(m-1) & \underset{(m-\frac{1}{2})}{\lessgtr} & \overset{\times}{\phi}(2m) & \underset{(m-\frac{1}{2})}{\lessgtr} & \hat{\phi}_a(m) \\ \check{\phi}_a(m) & < & & < & \check{\phi}_b(m) \end{array} \qquad \begin{array}{ccccc} \hat{\phi}_a(m) & < & & < & \hat{\phi}_b(m) \\ \check{\phi}_b(m) & \underset{m}{\lessgtr} & \overset{\times}{\phi}(2m+1) & \underset{m}{\lessgtr} & \check{\phi}_a(m+1) \end{array} \tag{B.19}$$

Adjusting for the intersection index i allows us to see that the intersection points are best classified by being either odd or even values of i

$$\begin{array}{ccccc} \hat{\phi}_b(\frac{i-2}{2}) & \underset{\frac{i-1}{2}}{\lessgtr} & \overset{\times}{\phi}(i) & \underset{\frac{i-1}{2}}{\lessgtr} & \hat{\phi}_a(\frac{i}{2}) \\ \check{\phi}_a(\frac{i}{2}) & < & & < & \check{\phi}_b(\frac{i}{2}) \end{array} \qquad \begin{array}{ccccc} \hat{\phi}_a(\frac{i}{2}) & < & & < & \hat{\phi}_b(\frac{i}{2}) \\ \check{\phi}_b(\frac{i}{2}) & \underset{\frac{i}{2}}{\lessgtr} & \overset{\times}{\phi}(i+1) & \underset{\frac{i}{2}}{\lessgtr} & \check{\phi}_a(\frac{i}{2}+1) \end{array} \tag{B.20}$$

$$\overbrace{\begin{array}{ccccc}\hat{\phi}_b(\frac{i-2}{2}) & & & & \hat{\phi}_a(\frac{i}{2}) \\ & \lll_{\frac{i-1}{2}} & \overset{\times}{\phi}(i) & \lll_{\frac{i-1}{2}} & \\ \check{\phi}_a(\frac{i}{2}) & < & & < & \check{\phi}_b(\frac{i}{2})\end{array}}^{i\in\{2\mathbb{N}\}} \qquad \overbrace{\begin{array}{ccccc}\hat{\phi}_a(\frac{i-1}{2}) & & < & & \hat{\phi}_b(\frac{i-1}{2}) \\ & & \overset{\times}{\phi}(i) & & \\ \check{\phi}_b(\frac{i-1}{2}) & \lll_{\frac{i-1}{2}} & & \lll_{\frac{i-1}{2}} & \check{\phi}_a(\frac{i+1}{2})\end{array}}^{i\in\{2\mathbb{N}+1\}}$$

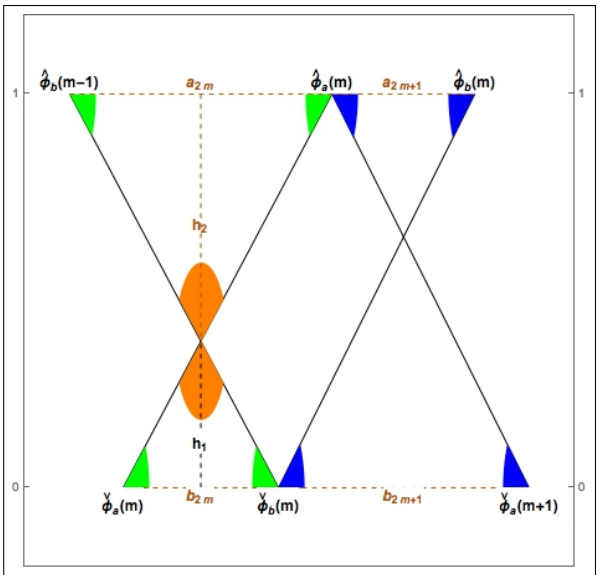
Recalling the definitions of the extrema functions

$$\begin{array}{llll}\check{\phi}_a\left(\dfrac{i}{2}\right)=\mathring{\phi}_a(i) & \check{\phi}_b\left(\dfrac{i}{2}\right)=\mathring{\phi}_b(i) & i\in\{0,2,4,6\cdots 2^{n-2}\} \\ \hat{\phi}_a\left(\dfrac{i-1}{2}\right)=\mathring{\phi}_a(i) & \hat{\phi}_b\left(\dfrac{i-1}{2}\right)=\mathring{\phi}_b(i) & i\in\{1,3,5,7\cdots 2^{n-2}-1\}\end{array}$$

the boundary relations become

$$\overbrace{\begin{array}{ccccc}\mathring{\phi}_b(i-1) & & & & \mathring{\phi}_a(i+1) \\ & \lll_{\frac{i-1}{2}} & \overset{\times}{\phi}(i) & \lll_{\frac{i-1}{2}} & \\ \mathring{\phi}_a(i) & < & & < & \mathring{\phi}_b(i)\end{array}}^{i\in\{2\mathbb{N}\}} \qquad \overbrace{\begin{array}{ccccc}\mathring{\phi}_a(i) & & < & & \mathring{\phi}_b(i) \\ & & \overset{\times}{\phi}(i) & & \\ \mathring{\phi}_b(i-1) & \lll_{\frac{i-1}{2}} & & \lll_{\frac{i-1}{2}} & \mathring{\phi}_a(i+1)\end{array}}^{i\in\{2\mathbb{N}+1\}}$$

We want to classify the points of intersection as greater or lesser than a tolerance $0 < \tau \leq 1$. In order to do this, we make a linear approximation to the perturbation between the extrema. There is one point of intersection between each of the maxima and corresponding minima. A bit of geometry allows us to write a classification criteria for each of the points of intersection.



$$\frac{h_1}{h_2} = \frac{b}{a} \quad h_1 = \frac{b}{a+b}$$
$$a_{2m} = \|\hat{\phi}_b(m-1) - \hat{\phi}_a(m)\|$$
$$b_{2m} = \check{\phi}_a(m) - \check{\phi}_b(m)$$
$$a_{2m+1} = \hat{\phi}_a(m) - \hat{\phi}_b(m)$$
$$b_{2m+1} = \|\check{\phi}_b(m) - \check{\phi}_a(m+1)\|$$

Defining a function for the degree of error at the point of intersection using the linear approximation from the four extrema surrounding the point , and generalizing to allow for different maximum errors α for channel a and β for channel b. The conditions on i and $\imath$ can be more compactly written by applying a condition to their product.

$$h(i,\imath,\alpha,\beta) = \begin{cases} \dfrac{\alpha\beta(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_b(i-\imath))}{\beta(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_a(i+\imath+1))+\alpha(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_b(i-\imath))} & i+\imath \in \{2\mathbb{N}\} \\ \dfrac{\alpha\beta(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_a(i+\imath+1))}{\beta(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_a(i+\imath+1))+\alpha(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_b(i-\imath))} & i+\imath \in \{2\mathbb{N}+1\} \end{cases} \tag{B.21}$$

allows the criteria for an acceptable perturbation to be written as $\tau \geq h(i,\imath)$. A more useful formulation is to specify an acceptable perturbation to each channel $\tau_\alpha = \frac{\tau}{\alpha}$ and $\tau_\beta = \frac{\tau}{\beta}$. We may as well choose $\tau = 1$ and use the channel specific values to control the acceptable perturbation.

$$h(i,\imath,\tau_\alpha,\tau_\beta) = \begin{cases} \dfrac{(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_b(i-\imath))}{\tau_\alpha(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_a(i+\imath+1))+\tau_\beta(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_b(i-\imath))} & i+\imath \in \{2\mathbb{N}\} \\ \dfrac{(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_a(i+\imath+1))}{\tau_\alpha(\mathring{\phi}_a(i+\imath)-\mathring{\phi}_a(i+\imath+1))+\tau_\beta(\mathring{\phi}_b(i-\imath-1)-\mathring{\phi}_b(i-\imath))} & i+\imath \in \{2\mathbb{N}+1\} \end{cases} \tag{B.22}$$

$$1 \geq h(i,\imath,\tau_\alpha,\tau_\beta) \tag{B.23}$$

Given a desired angle of rotation θ; a data type, which determines n; and the acceptable channel perturbations, specified by τ_α and τ_β we can find the nearest angle which minimises the quantisation error and a close estimate of the maximum error which can occur.

$$\iota = \left\lfloor 7\,\bar{\mathbf{i}} - \sqrt{i^2 - 2i\,\bar{\mathbf{i}} + 49\,\bar{\mathbf{i}}^2} \right\rfloor \qquad \bar{\mathbf{i}} = 2^{n-3} \tag{B.24}$$

$$i = \frac{\bar{\mathbf{i}}\tan(\theta_1)\left(\sqrt{3}\tan(\theta_1)+7\right)}{\tan(\theta_1)+\sqrt{3}} \quad \text{where} \quad \theta_1 = \theta \bmod \frac{\pi}{6} \tag{B.25}$$

Starting from the value i the algorithm searches for $\tilde{i}$ the value nearest to i which satisfies the condition $1 \geq h(i,\iota,\tau_\alpha,\tau_\beta)$ (B.22).

The algorithm then finds a starting value for the index (B.15) and returns the closest value which satisfies the the condition $1 \geq h(i,\iota,\tau_\alpha,\tau_\beta)$ (B.22):

The region ι for a given value of i is found and the appropriate values (B.17) for the extrema (B.12) are evaluated in the function which finds the perturbation at the index (B.21) and the intercept position $\overset{\times}{\phi}(i)$ is found with algorithm 5.

Algorithm 5 A function which returns the angular position of compromise between the perturbations to the channels.

function $\overset{\times}{\phi}(i,\tau_\alpha,\tau_\beta,n)$ ▷ i integer index of the intercept

$\bar{\mathbf{i}} \leftarrow 2^{n-3}$; $\iota(i,n) \leftarrow \left\lfloor 7\,\bar{\mathbf{i}} - \sqrt{i^2 - 2i\bar{\mathbf{i}} + 49\,\bar{\mathbf{i}}^2} \right\rfloor$ ▷ n the source bit depth

$\mathring{\phi}_b(\mathbf{l}) = \arctan\left(\frac{\mathbf{l}2^{2-n}}{\sqrt{3}}\right)$; $\mathring{\phi}_a(\mathbf{l}) = \arctan\left(\frac{\mathbf{l}\sqrt{3}}{2^n - \mathbf{l}}\right)$

if $i+\iota \bmod 2 = 0$ **then** ▷ $i+\iota$ is even

$\hat{\phi}_a \leftarrow \mathring{\phi}_a(i+\iota+1)$; $\check{\phi}_a \leftarrow \mathring{\phi}_a(i+\iota)$

$\hat{\phi}_b \leftarrow \mathring{\phi}_b(i-\iota-1)$; $\check{\phi}_b \leftarrow \mathring{\phi}_b(i-\iota)$

else

$\hat{\phi}_a \leftarrow \mathring{\phi}_a(i+\iota)$; $\check{\phi}_a \leftarrow \mathring{\phi}_a(i+\iota+1)$

$\hat{\phi}_b \leftarrow \mathring{\phi}_b(i-\iota)$; $\check{\phi}_b \leftarrow \mathring{\phi}_b(i-\iota-1)$

end if

$$\overset{\times}{\phi}(i) \leftarrow \left\{ \frac{\tau_\alpha\check{\phi}_b(\check{\phi}_a-\hat{\phi}_a)+\tau_\beta\check{\phi}_a(\hat{\phi}_b-\check{\phi}_b)}{\tau_\alpha(\check{\phi}_a-\hat{\phi}_a)+\tau_\beta(\hat{\phi}_b-\check{\phi}_b)}, \frac{(\check{\phi}_a-\check{\phi}_b)}{\tau_\alpha(\check{\phi}_a-\hat{\phi}_a)+\tau_\beta(\hat{\phi}_b-\check{\phi}_b)} \right\}$$

Return $\overset{\times}{\phi}(i)$ ▷ $\overset{\times}{\phi}(i)$ {angle, maximum perturbation}

end function

To find the angular value closest to θ which produces a perturbation less than the tolerance τ, algorithm 6 finds the closest value which satisfies the tolerance in the positive and negative directions and returns the nearest one to the requested value of θ. Algorithm 6 is

guaranteed to find a value which satisfies the tolerance in each direction within a $\frac{\pi}{6}$ region because the perturbation is zero at the ends of the $\frac{\pi}{6}$ region.

Algorithm 6 Suggest a new value for θ

Require: τ_α, τ_β the maximum allowed perturbation to the channels
θ the requested angle;

Ensure: ϑ the suggested value for θ and h the predicted maximum perturbation.

$\delta\theta \leftarrow \theta \bmod \frac{\pi}{6}$; $\Theta \leftarrow \theta - \delta\theta$; $\bar{\mathbf{i}} \leftarrow 2^{n-3}$;

$i_\ominus \leftarrow i_\oplus \leftarrow \frac{\bar{\mathbf{i}}\tan(\delta\theta)\left(\sqrt{3}\tan(\delta\theta)+7\right)}{\tan(\delta\theta)+\sqrt{3}}$; $\{\vartheta_\ominus, h_\ominus\} \leftarrow \{\vartheta_\oplus, h_\oplus\} \leftarrow \overset{\times}{\phi}\left(i, \tau_\alpha, \tau_\beta, n\right)$

while $h_\oplus > 1$ **do**
  $i_\oplus ++$
  $\{\vartheta_\oplus, h_\oplus\} \leftarrow \overset{\times}{\phi}\left(i_\oplus, \tau_\alpha, \tau_\beta, n\right)$
end while
while $h_\ominus > 1$ **do**
  $i_\ominus --$
  $\{\vartheta_\ominus, h_\ominus\} \leftarrow \overset{\times}{\phi}\left(i_\ominus, \tau_\alpha, \tau_\beta, n\right)$
end while
if $\vartheta_\oplus - \delta\theta < \delta\theta - \vartheta_\ominus$ **then**
  $\{\vartheta, h\} = \{\Theta + \vartheta_\oplus, h_\oplus\}$
else
  $\{\vartheta, h\} = \{\Theta + \vartheta_\ominus, h_\ominus\}$
end if
Return $\{\vartheta, h\}$

This allows us to construct an integer rotation matrix which is sufficiently close to the floating point rotation matrix that it introduces an error in the rotated pixel values no greater than τ_α and τ_β in the two chromatic channels. The only question left is how onerous is the restriction on the choice of angle. The algorithm would be useless if the suggested angle is very different from the requested angle. To investigate this lets look at the special case where no error is allowed. The criteria $\tau_\alpha = \tau_\beta = ½$ gives an integer rotation matrix which produces the same effect as the floating point rotation matrix. It also selects from either the odd values or the even values of i, because if $h(i, \imath, \tau_\alpha, \tau_\beta) < ½$ is true, then $h(i+1, n, , \tau_\alpha, \tau_\beta) < ½$ is false and vice versa. There are 2^{n-2} points of intersection and 2^{n-3} which satisfy the criteria $\tau = \frac{1}{2}$ in each $\frac{\pi}{6}$ region.

For a 24 bit color image n is 8, so iR $\in \{-2^{8-2} \cdots 2^{8-2}\}$, which gives $2^{8-3} - 1$ values

in each $\frac{\pi}{6}$ region, producing a maximum perturbation of less than ½. In total, there are $12(2^{8-3}-1) = 372$ possible values for θ which produces a maximum perturbation of less than ½. This would allow the destination pixel values to be expressed in 8-bit numbers without error using a transformation matrix expressed in 6-bit signed integers. The statistics performed to determine θ are not demanding enough to justify rejecting a suggested value of θ within 1°of the requested value. It is, however, important to know the value actually used in the algorithm, which is why the mechanism for adjusting θ is separate from the color-space algorithm and is under the control of the programmer.

Appendix C

Preservation of Color Information

The goal of the algorithm is to preserve all the information captured by the camera which relates to skin whilst discarding as much of the irrelevant information as possible. Given that edges and features often present as shadows and highlights, all the information captured in terms of luminosity will be regarded as relevant information, at least as far as the manipulation of individual pixel values is concerned. Considering the chromatic information, the importance of the pixel value will be directly determined by a Gaussian distribution.

Knowing the range of values produced by the rotation allows us to scale the transformation to fit into the range of the destination data type. If we have RGB pixel values in a given machine data type, the amount of information contained in each of those channels is equal to the number of values accessible in that data type. For example: for 8-bit, unsigned integers, there are 256 possible values. After a rotation, we are interested in the amount of information which lies along the new axes. This is found simply by multiplying the range of the source data type by the length of the new axes found for the unit cube. To preserve all the information captured, we would therefore have to use a larger data type to store the new values. We are, however, only interested in a small region in the chromatic space. The question is, then, how to preserve the relevant information in a way consistent with the significance indicated by the aforementioned Gaussian distribution.

None of the rotated axes have lengths less than 1 for the unit RGB cube. For this reason we've written re-distribution functions which perform any necessary type conversion whilst preserving the information in a controlled way; we can keep the information where it's needed and discard it where it's irrelevant. So although this is strictly beyond the normal meaning of a color space conversion, it is addressing a connected issue and belongs in the conversion. In terms of optimization, it is also the most efficient place in the code in which to perform this adjustment, allowing us to — for the sake of example — discard the details

of the colors of a duck's feathers whilst keeping the hues and tones of human skin.

We can use a function to redistribute the information contained on the longer axis onto the shorter axis, which can be expressed in the discrete representation of that axis necessitated by internal integer data types. There are three ways in which to implement the re-distribution functions:

C.0.1 Partition

The most straightforward re-distribution method is to simply preserve the information in a 1-to-1 fashion within a region. The region can be defined in terms of the distribution Gaussian by specifying a significance level in terms of the variance or the standard deviation.

C.0.2 Linear

A slightly more sophisticated method is to use a linear re-distribution. A linear distribution is equivalent to partitioning given a unit gradient. However, a linear re-distribution function allows for the possibility of data compression. So then, in the case where the region of interest contains more information than can be expressed in the destination data type, a linear distribution function allows an even compression of the information from source to destination. **?**

C.0.3 ERF (Gaussian Error Function)

The integral of the cumulative Gaussian (i.e. Error Function) allows the re-distribution of the information on the axis in a way which selectively preserves the information about a point on the axis (i.e. the mean of the Gaussian), and then progressively discard the information as it falls into the tails of the Gaussian. So, it provides a non-linear distribution of the information. The Gaussian can be seen as describing our interest in the information contained along the axis, so it's logical to use the error function to redistribute the information. The disadvantage of this is simply the computational effort involved in generating the error function.

The error function distribution is mathematically correct, being directly related to the Gaussian fit. Computationally, there are two considerations: the numerical representation, and performance. The discrete representation of the numerics means that — for a significant number of possible distributions — distributing using the error function has little to no advantage (or indeed difference) from using a linear distribution.

Considering the preservation of information captured, mappings with a gradient greater than 1 are undesirable because they preserve all the information whilst being informatically wasteful in that there are functionally inaccessible discrete values in the destination range. Our stated aim is to preserve the information in the image pertaining to human skin; unevenly distributing this information across a discrete data type is not only wasteful in terms of memory, but also of processing resources because subsequent processing routines will treat the data as if it has a higher fidelity than it actually does.

To construct a distribution function, we first need to describe the relationship of the error function to the Gaussian fit, and then produce a function with the appropriate range and domain. For a Gaussian fit with an amplitude A, a mean of μ, and a standard deviation of σ, where μ and x lie in a source range xRange from xMin to xMax. The cumulative distribution is found by integrating from xMin to the point x, as can be seen in (C.1):

$$\int_{\text{xMin}}^{\text{x}} A\frac{e^{-\frac{(t-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}dt = \frac{1}{2}A\left(\text{erf}\left(\frac{\text{x}-\mu}{\sqrt{2}\sigma}\right) - \text{erf}\left(\frac{\text{xMin}-\mu}{\sqrt{2}\sigma}\right)\right) \tag{C.1}$$

The Gaussian distribution and the cumulative distribution are shown in Figure C.1 for some values chosen for illustrative purposes. All that is required now is to fix the range yMin to yMax for the domain xMin to xMax.

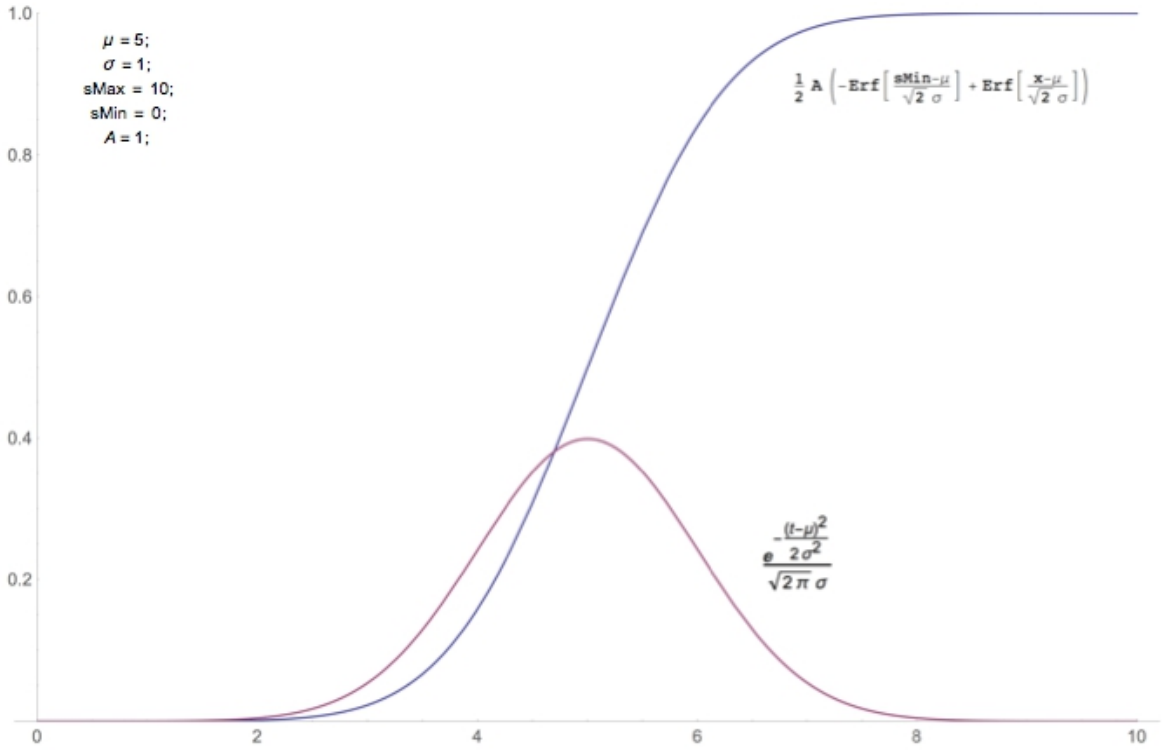
First, we determine the maximum value taken in the domain. This is simply found by evaluating the function at xMax. It should be noted that — if the Gaussian distribution is well contained in the source domain — the maximum value should be equal to the amplitude. For the sake of simplicity, we'll ignore the amplitude of the fitted Gaussian found previously as it is not relevant to the design of the re-distribution function. So, to fix the range of the distribution function, we first scale to the range $0:1$ by simply dividing through by the maximum value, and then re-scale to the destination range yRange $=$ yMax $-$ yMin and shift by yMin.

$$dis(x) = \frac{(\text{yRange})\left(\text{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right) - \text{erf}\left(\frac{\text{xMin}-\mu}{\sqrt{2}\sigma}\right)\right)}{\text{erf}\left(\frac{\text{xMax}-\mu}{\sqrt{2}\sigma}\right) - \text{erf}\left(\frac{\text{xMin}-\mu}{\sqrt{2}\sigma}\right)} + \text{yMin} \tag{C.2}$$

C.0.4 Efficiently Implementing the Distribution Function

Mathematically, the ERF distribution function (C.2) achieves all the stated objectives. However, on a device we are dealing with discrete numerics and limited processing power, so further analysis is required. Where we're using a discrete domain and range, the distribu-

Fig. C.1 Error function.



tion is usefully divided into three characteristic behaviours: where it is constant, where it preserves all the information, and where it selectively preserves information. Looking at the distribution, this divides the source domain into five regions: two where it is effectively constant, two where it is selective, and one region around the mean where it preserves all the information. In order to design an efficient algorithm, it is useful to identify the boundaries of these five regions.

C.0.4.1 The Region Which Discards All Information

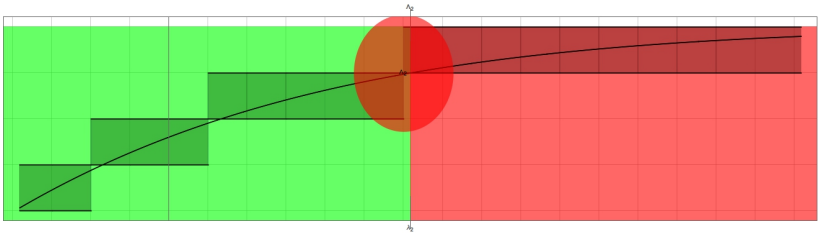


Fig. C.2 The region which discards all information beyond Λ_2 is shown above. The shaded grid squares show the value actually taken by the discrete distribution.

First, we need to identify where the distribution is effectively constant. This can be found

by solving the following equation in the region and domain $0 \leq x \leq 1$ and then generalized to the specific discrete numerics:

$$\frac{\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)+\operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)}{\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right)}=\mathrm{dL} \quad \text{where} \quad \mathrm{dL}=\left\{\frac{1}{\mathrm{yRange}},1-\frac{1}{\mathrm{yRange}}\right\} \tag{C.3}$$

The solution is found for the source domain in the range $0:1$ as:

$$x=\sqrt{2}\sigma\operatorname{erf}^{-1}\left((\mathrm{dL}-1)\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)-\mathrm{dL}\operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right)\right)+\mu \tag{C.4}$$

The boundaries of the regions $x<\Lambda_1$ and $x>\Lambda_2$ for $x\in\{\mathrm{tMin}\cdots\mathrm{tMax}\}$ or $x<\lambda_1$ and $x>\lambda_2$ for $x\in\{0\cdots1\}$ can be written using the following helpful constants of the distribution.

$$\Sigma^{-}=\operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right) \qquad \Sigma^{+}=\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right) \qquad \mathrm{dL}=\frac{1}{\mathrm{yRange}} \qquad \kappa=\frac{\mathrm{yRange}}{\mathrm{tRange}} \tag{C.5}$$

$$\begin{aligned}
&\lambda_1=\sigma\sqrt{2}\operatorname{erf}^{-1}\left((\mathrm{dL}-1)\Sigma^{+}-\mathrm{dL}\,\Sigma^{-}\right)+\mu \quad && \lambda_2=\sigma\sqrt{2}\operatorname{erf}^{-1}\left((\mathrm{dL}-1)\Sigma^{-}-\mathrm{dL}\,\Sigma^{+}\right)+\mu\\
&\Lambda_1=\mathrm{tMin}+\mathrm{tRange}\left(\lambda_1(\mu,\sigma)\right) && \Lambda_2=\mathrm{tMin}+\mathrm{tRange}\left(\lambda_2(\mu,\sigma)\right)
\end{aligned} \tag{C.6}$$

C.0.4.2 The Region Which Keeps All Information

To find the region where all the information in the source domain is preserved, we differentiate the distribution and solve for where the gradient is equal to the destination range over the source range. This corresponds to the point at which a unit change in the source produces a unit change in the destination range:

$$\frac{\sqrt{\frac{2}{\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\left(\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right)\right)}=\frac{\mathrm{yRange}}{\mathrm{tRange}} \tag{C.7}$$

Rearranging for x, we find:

$$x = \mu \pm \sigma \sqrt{-2\log\left(\sigma\left(\operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right)\right)\right) + 2\log\left(\frac{\text{tRange}}{\text{yRange}}\right) + \log\left(\frac{2}{\pi}\right)} \tag{C.8}$$

The boundaries of the region $\Omega_1 < x < \Omega_2$ for $x \in \{\text{tMin} \cdots \text{tMax}\}$ and $\omega_1 < x < \omega_2$ for $x \in \{0 \cdots 1\}$ can be written using the following helpful constants of the distribution.

$$\Sigma^- = \operatorname{erf}\left(\frac{\mu-1}{\sqrt{2}\sigma}\right) \qquad \Sigma^+ = \operatorname{erf}\left(\frac{\mu}{\sqrt{2}\sigma}\right) \qquad \kappa = \frac{\text{yRange}}{\text{tRange}} \tag{C.9}$$

$$w(\mu,\sigma) = \sigma\sqrt{\log\left(\frac{2}{\pi}\right) - 2\log\left(\kappa\sigma\left(\Sigma^+ - \Sigma^-\right)\right)} \tag{C.10}$$

The equations are found in the unit source domain $0:1$. It is a simple matter to scale and shift these values to give the points in a more general source domain.

$$\begin{aligned} \omega_1(\mu,\sigma) &= \mu - w(\mu,\sigma) & \omega_2(\mu,\sigma) &= \mu + w(\mu,\sigma) \\ \Omega_1(\mu,\sigma) &= \text{tMin} + \text{tRange}\left(\mu - w(\mu,\sigma)\right) & \Omega_2(\mu,\sigma) &= \text{tMin} + \text{tRange}\left(\mu + w(\mu,\sigma)\right) \end{aligned} \tag{C.11}$$

One refinement can be made to these values by recognizing that the discrete distribution extends the effectively linear region past the analytic solution by rounding the values. This can be seen in C.3, where the shaded squares are the rounded values. The extended region boundary Ωp was found by numerically solving

$$\lfloor \text{dis}\left(\Omega_2\right) \rfloor + \left(x - \Omega_2\right) = \text{dis}(x) \tag{C.12}$$

In the C++ code, whilst numerical routines were used MatLab and Mathematica to perform the analysis, the extended region boundary point was found by 'walking' along the distribution from the analytic point Ω_2 until divergence from linear behaviour became apparent. This is regarded as a simpler solution, not requiring the use of numerical library routines, and proved to be a quick and elegant solution for the C++ implementation. The

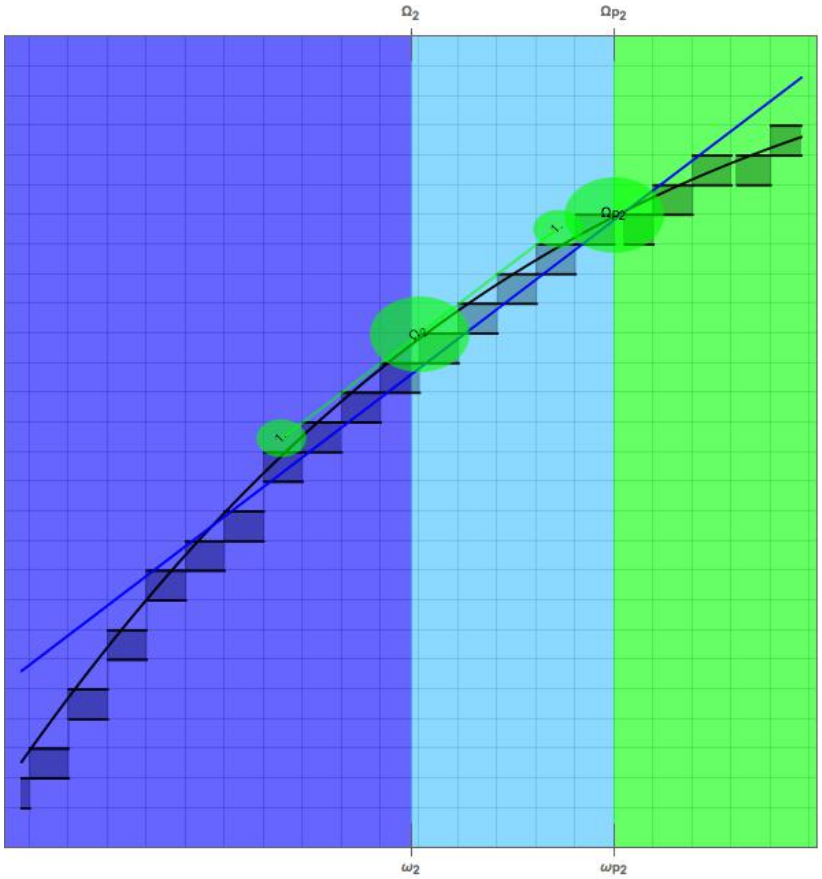


Fig. C.3 The region which preserves all information extends beyond the analytic region due to the rounding involved in the discretization. The shaded grid squares show the value actually taken by the discrete distribution. The cord is the tangent to the distribution curve shifted to the next discrete unit below the curve at Ω_2. The point of intersection is the extended boundary at which the distribution begins to discard information.

extended boundaries Ωp_1 and Ωp_2 are then defined by

$$\lceil \mathrm{dis}\,(\Omega_1) \rceil - \Omega_1 = \mathrm{dis}\,(\Omega p_1) - \Omega p_1 \quad \lfloor \mathrm{dis}\,(\Omega_2) \rfloor - \Omega_2 = \mathrm{dis}\,(\Omega p_2) - \Omega p_2 \tag{C.13}$$

C.0.4.3 The Compression Ratio

There is one final value of interest to the development of the algorithm, which is the gradient at the mean. The reason this is of interest is because we're trying to compress the relevant data as much as possible. If the destination region is small (i.e. the destination machine type is smaller than the source type), then the gradient at the mean allows us to assess the fidelity required of the source type. If it weren't for the fact that the source is the result of a rotation

transformation, then there would be little purpose in assessing this value. However, it is entirely possible that the lengthening of the axes resulting from the rotation is insignificant for the desired destination type; there's no point preserving information during the rotation which is then discarded by the re-distribution. The gradient is given by

$$\Delta(\mu,\sigma) = \kappa\delta(\mu,\sigma) \quad \delta(\mu,\sigma) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}\,(\Sigma^{+} - \Sigma^{-})} \tag{C.14}$$

The compression ratio is at most one-to-one, therefore $\kappa <= 1$ and the gradient in the unit space must always be greater than one $1 \leq \delta(\mu,\sigma)$. so $\kappa \leq \Delta(\mu,\sigma) \leq \delta(\mu,\sigma)$.

The required fidelity in the source domain can be found using Δ in the sense that the correspondence between one information step in the source must produce a step of Δ in the destination type. For the algorithm evaluating the maximum gradient Δ allows us to be sure that Ω — the region on the x axis where all the information is to be preserved — exists if $\Delta > 1$ or tells us that the x axis can be shortened if $\Delta < 1$. In the algorithm Δ is used to find an appropriate working data type for the rotated color space and to define the axis scaling for the rotation matrix.

We need to consider the requested compression of information alongside the spread of information caused by the rotation, and the desired focus on the specific region of interest dictated by the statistics. Each axis in the color space is to be represented by a discrete set of numbers. The size of these sets dictates the discretization of the axis, and the ratios between them indicates the spread or compression of the information they contain. We assume that the RGB axes are each discretized to the same extent, each containing srcRange values. After the application of the un-normalized rotational transformation, the axes contain differing numbers of values given by $\text{tRange}_1 = \text{srcRange}\,\sqrt{3}$, $\text{tRange}_2 = \text{srcRange}\,L_2(\theta)$ and $\text{tRange}_3 = \text{srcRange}\,L_3(\theta)$. These axis lengths preserve all the information contained in the source color space, and so are the maximum length the axis should take. The minumum length the axis may take is where the information is lost evenly throuought the axis, and corresponds to an axis length equal to the destination axis length tRange = dstRange.

We can now write an algorithm which determines the necessary scaling for the axes, and whether truncation of the extreme values is significant. As this will alter tRange we fix the values of κ and Δ to be those for the working range $\text{tRange} = \vec{L}(\theta)\,\text{srcRange}$ which preserves all information. With this value the constants are

$$K = \frac{\text{dstRange}}{\text{srcRange}} \qquad \kappa(\theta) = \frac{K}{\mathbf{L}(\theta)} \qquad \Delta(\mu,\sigma) = \frac{K}{\mathbf{L}(\theta)}\delta(\mu,\sigma) \tag{C.15}$$

The length of the axis tRange after rescaling should be

$$\begin{aligned} \text{tRange}(\theta,\mu,\sigma) &= \min\left\{\frac{K}{\mathbf{L}(\theta)}\delta(\mu,\sigma),1\right\}\vec{L}(\theta)\,\text{srcRange} \\ &= \min\left\{K\delta(\mu,\sigma)\,,\vec{L}(\theta)\right\}\text{srcRange} \end{aligned} \tag{C.16}$$

The scaling srcRange comes from the source pixel values, the remaining terms translates simply into a rotation matrix scaling

$$\mathbf{R}(\theta) = \min\left\{K\delta(\mu,\sigma)\,,\vec{L}(\theta)\right\}\otimes\vec{S}(\theta)\otimes\mathbf{fRO}\left[\mathbf{qRe}(\theta,n)\right]$$

This satisfies the requirements placed on the gradient $\Delta(\mu,\sigma) > 1$ because: if we substitute for tRange in the definition for the gradient C.14

$$\Delta(\mu,\sigma) = \max\left\{\frac{K}{\mathbf{L}(\theta)}\delta(\mu,\sigma),1\right\} \tag{C.17}$$

And the compression ratio κ is also explicitly restricted to being at most one and now is no longer defined in terms of tRange.

$$\kappa(\theta) = \max\left\{\frac{1}{\delta(\mu,\sigma)},\frac{K}{\mathbf{L}(\theta)}\right\} \tag{C.18}$$

C.0.4.4 A Piecewise Approximation to the ERF Distribution

We now have equations which give us the four points in the source domain which mark the boundaries of the five characteristic regions. We can now use them to define a piecewise function which uses the computationally problematic error function based distribution as little as possible

$$pDis(x) = \begin{cases} \text{dstMin} & x \le \Lambda_1 \\ \text{dis}(x) & \Lambda_1 < x < \Omega p_1 \\ x - \Omega p_1 + \text{dis}(\Omega p_1) & \Omega p_1 \le x \le \Omega p_2 \\ \text{dis}(x) + \Omega p_2 - \Omega p_1 - \text{dis}(\Omega p_2) + \text{dis}(\Omega p_1) & \Omega p_2 < x < \Lambda_2 \\ \text{dis}(\Lambda_2) + \Omega p_2 - \Omega p_1 - \text{dis}(\Omega p_2) + \text{dis}(\Omega p_1) & x \ge \Lambda_2 \end{cases} \tag{C.19}$$

All three distribution techniques described earlier (C.0.1 , C.0.2 , C.0.3) are special cases of this distribution. When the distribution has a very large variance the piecewise dis-

tribution can be simplified as a linear distribution Fig C.4. When the distribution has a very small variance a partitioning is more appropriate Fig C.5. The most interesting distributions, however, are the ones which require the use of the piecewise distribution Fig C.6.

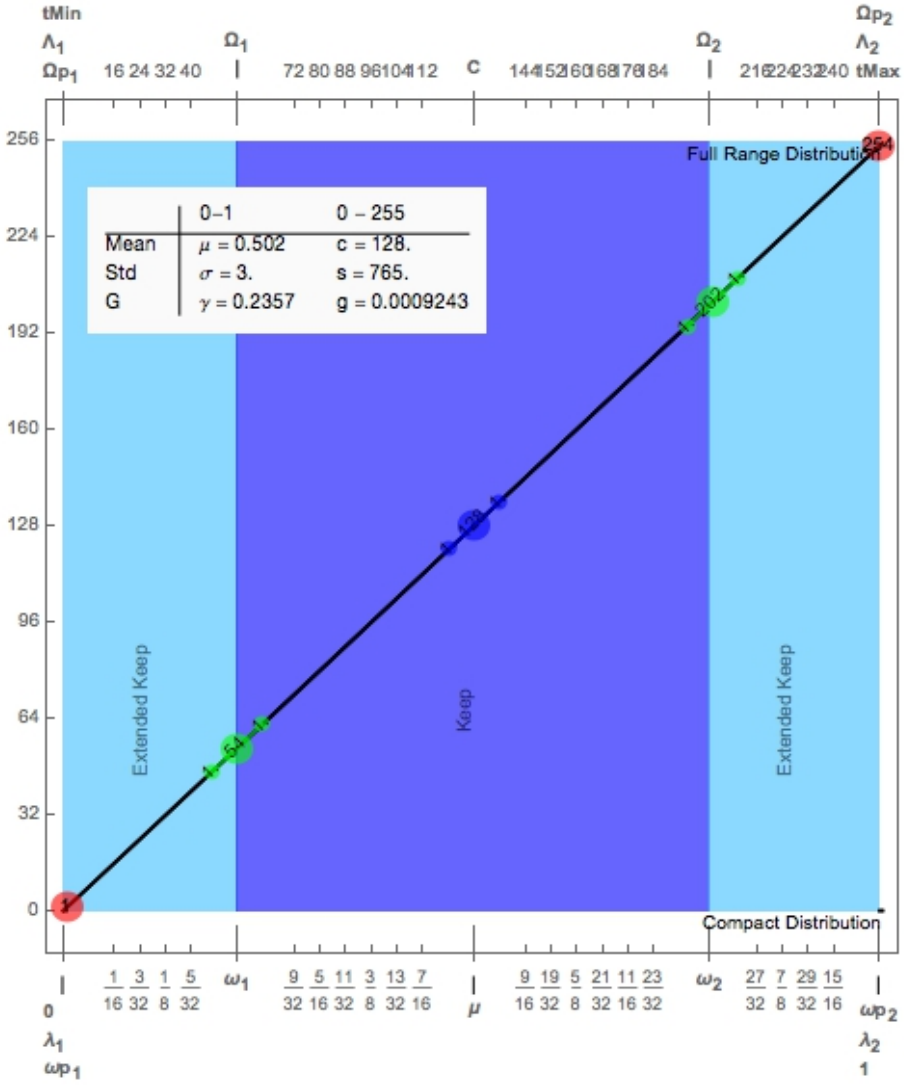


Fig. C.4 Here the code will decide to use a linear re-distribution.

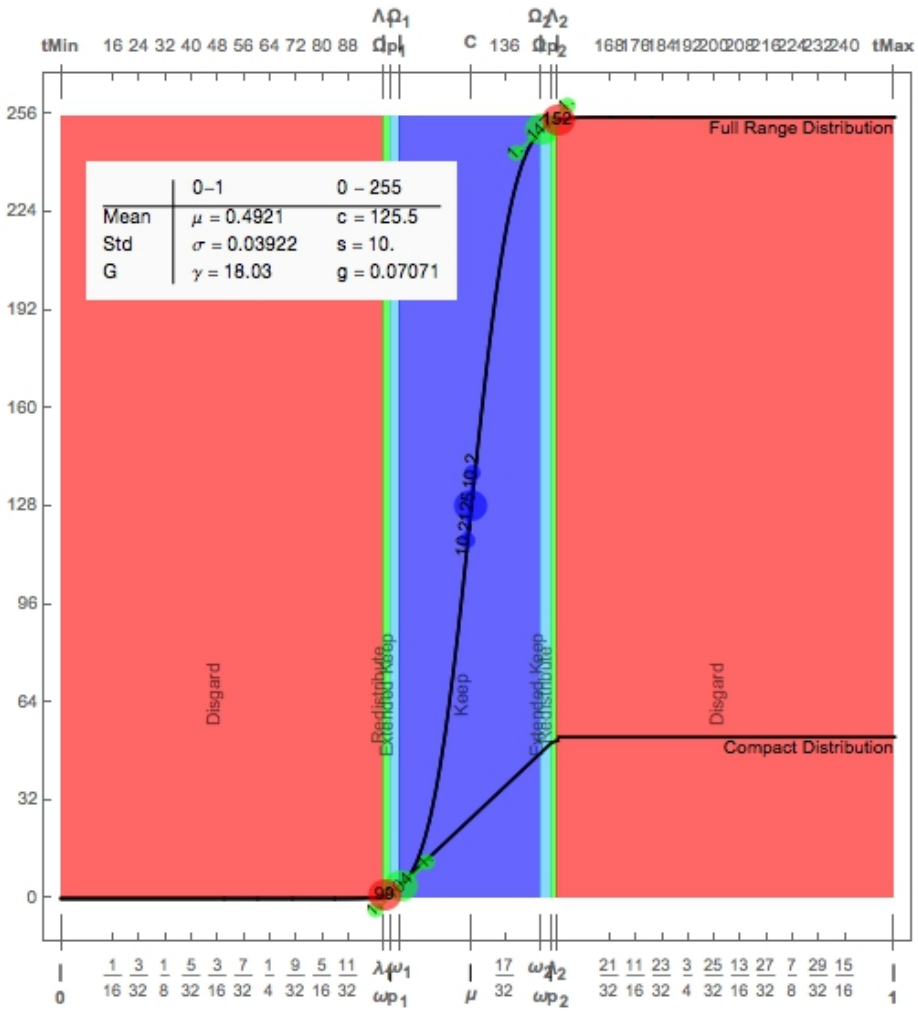


Fig. C.5 With these values the code will likely decide to use partitioning if the tolerance $\tau_{\text{partitioning}}$is greater than $\Lambda_2 - \Omega p_2$.

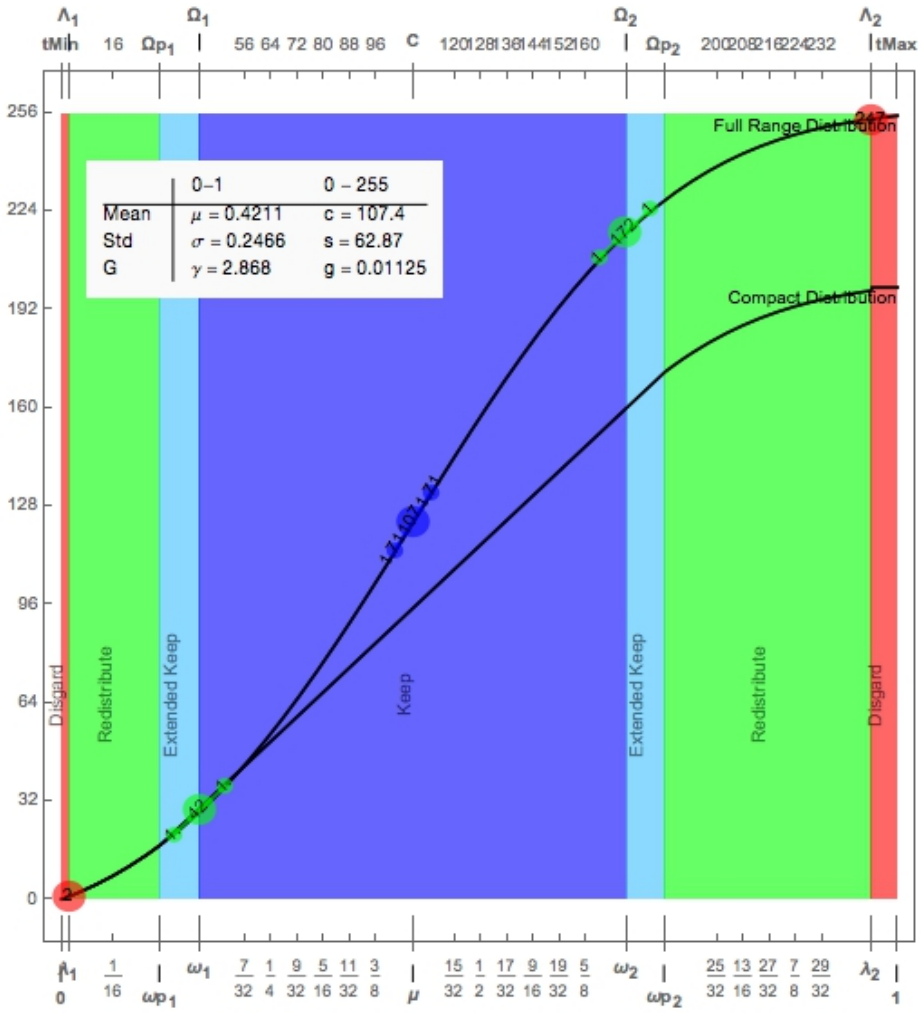


Fig. C.6 With these values the code will decide to use a piecewise ERF based distribution. (Labeled "Compact Distribution" in the figure.)

Appendix D

Chapter 4 Colorspace Methods

D.1 Implementing the Color Space

In the previous two chapters, we have been approaching the color space design problem mathematically. Here, we will outline the practical implementation of the color space algorithm on the mobile device.

The first and most pressing consideration is limited processing power; while our mathematical approach to the algorithm design is significantly faster and more efficient than the more typical color space algorithms — as will be discussed in the following chapter — minimizing the number of clock cycles spent on continuous processes is necessary in order to ensure efficient operation. In the case of redistribution, the integer matrix transform is performed first, which puts the pixel values in the quantized working type *qRsRange* described in Chapter 2, then pixel classification is performed before any of the chromatic information is redistributed as illustrated in Figure D.1.

The pixel classification is performed before the redistribution because it excludes processing pixels which are of no interest based on both chromatic channels, whereas for redistribution one channel may clearly lie outside of the discard region λ while the other could be kept or redistributed. With the pixel classifier, if one channel lies outside the discard region, the other is also of no interest since the pixel value is outside of the partitioning region, thereby avoiding unnecessary processing time spent on redistribution. The classification is straightforward within the limits on the channel values, so it isn't sophisticated like an elliptical partitioning or a probabilistic Gaussian-based thresholding; it is simply a quick way of excluding the majority of the pixel values which aren't of interest.

Once the pixels have been classified, the redistribution is performed — which requires further processing — or we assign a pixel value from a small set of possibilities and skip

onto the next pixel value. The redistribution function is applied to the working type *tRange*, while the comparison and classification is done in *qRsRange*. This is because the information in *tRange* is compressed to avoid wasting space, and as a consequence of this it does not occupy the full data type. *qRsRange*, on the other hand, being the natural range multiplied by the quantization matrix qR, does occupy the full data type. Since the operation is simply a comparison between numbers, it is of no advantage to compress the information so tightly. Additionally, since *qRsRange* is a signed range, one could look at the bit indicating the sign and immediately tell whether or not a given value will be discarded. Once scaled to *tRange*, the redistribution function is applied — which has a source range of *tRange* and a range *dRange* of the destination type — as outlined in Section C in Chapter 2.

It should be noted that the region between the extended keep region ωp and the discard region λ is not especially large; for even a 5σ result, it's only tens of values wide. As such, we use a lookup table for the redistribution in our actual implementation, thus further reducing the processing overhead. The C++ implementation has a lookup table threshold set to 64, a quarter of the source data type range. If the redistribution region is larger than that, the error function approximation from Chapter 2 is used. This is mainly for future-proofing the implementation; given the limited size of the region, we fully expect to use a lookup table for all practical cases.

As another future concession, the facility to add pixel-based functions to the color space transform, such as the skin probability function mentioned at the end of the previous chapter, is allowed. The color space transform is a threaded process, so there is no way of knowing that the last pixel value that was processed by any given thread was an adjacent pixel, as the parallelization is handled by OpenCV's universal threading library implementations. Pixel-based functions, however, are per-pixel processes, and they come out as separate resulting channels, so they can be added to the color space transform without causing any conflicts.

D.1.1 The White-Out Black-Out Algorithm

We need to find the point at which a chromatic value begins to suffer from white-out and black-out. This is done by taking the corresponding RGB value for the chromatic point and illuminating and deluminating the point by shifting the channel values by the same amount, noting when each channel value becomes fully saturated or desaturated. This is done in the unit space so the results can be scaled to any working range. The algorithm for finding the theoretical values is given in Algorithm 7 below.

Algorithm 7 The White-Out Black-Out Algorithm

Require: Ca, Cb The chromatic values

$iLCaCb$ the transform to LCaCb from RGB;

$LCaCb$ the transform to RGB from LCaCb

Ensure: The luminosity at which black out L_{BO} and white out L_{WO} occur.

$Ca_{min} \leq Ca \leq Ca_{max}$ The chromatic range which could be suffering from white-

$Cb_{min} \leq Cb \leq Cb_{max}$ out or black-out

$pnt^{RGB} \leftarrow iLCaCb(0.5, Ca, Cb)$ ▷ Put the point in RGB space

$O \leftarrow Order(pnt^{RGB}, \#1 > \#2)$ ▷ The order of the RGB channels from largest to smallest

$\Delta WO_1 \leftarrow 1 - pnt^{RGB}_{O(1)}$ ▷ Iluminate to make the largest channel value saturated

$pnt^{(RGB,WO,1)}_{O(1)} \leftarrow 1$ ▷ The point where a channel is saturated

$pnt^{(RGB,WO,1)}_{O(2)} \leftarrow pnt^{RGB}_{O(2)} + \Delta WO_1$

$pnt^{(RGB,WO,1)}_{O(3)} \leftarrow pnt^{RGB}_{O(3)} + \Delta WO_1$

$\Delta WO_2 \leftarrow 1 - pnt^{RGB}_{O(2)}$ ▷ Iluminate to make the second largest channel value saturated

$pnt^{(RGB,WO,2)}_{O(1)} \leftarrow 1$ ▷ The point where two channels are saturated

$pnt^{(RGB,WO,2)}_{O(2)} \leftarrow 1$

$pnt^{(RGB,WO,2)}_{O(3)} \leftarrow pnt^{RGB}_{O(3)} + \Delta WO_2$

$\Delta BO_1 \leftarrow pnt^{RGB}_{O(3)}$ ▷ Deluminate by the smallest channel value

$pnt^{(RGB,BO,1)}_{O(1)} \leftarrow pnt^{RGB}_{O(1)} - \Delta BO_1$ ▷ The point where a channel is desaturated

$pnt^{(RGB,BO,1)}_{O(2)} \leftarrow pnt^{RGB}_{O(2)} - \Delta BO_1$

$pnt^{(RGB,BO,1)}_{O(3)} \leftarrow 0$

$\Delta BO_2 \leftarrow pnt^{RGB}_{O(2)}$ ▷ Deluminate by the second smallest channel value

$pnt^{(RGB,BO,2)}_{O(1)} \leftarrow pnt^{RGB}_{O(1)} - \Delta BO_2$ ▷ The point where two channels are desaturated

$pnt^{(RGB,BO,2)}_{O(2)} \leftarrow 0$

$pnt^{(RGB,BO,2)}_{O(3)} \leftarrow 0$

$pnt^{(WO,i)} \leftarrow LCaCb(pnt^{(WO,i)})$ ▷ The white-out points in LCaCb color space

$pnt^{(BO,i)} \leftarrow LCaCb(pnt^{(BO,i)})$ ▷ The black-out points in LCaCb color space

$L_{WO} \leftarrow pnt^{(WO,1)}_1$

$L_{BO} \leftarrow pnt^{(BO,1)}_1$

$Ca_{min} \leftarrow Min(pnt^{(WO,1)}_2, pnt^{(WO,2)}_2, pnt^{(BO,1)}_2, pnt^{(BO,2)}_2)$

$Ca_{max} \leftarrow Max(pnt^{(WO,1)}_2, pnt^{(WO,2)}_2, pnt^{(BO,1)}_2, pnt^{(BO,2)}_2)$

$Cb_{min} \leftarrow Min(pnt^{(WO,1)}_3, pnt^{(WO,2)}_3, pnt^{(BO,1)}_3, pnt^{(BO,2)}_3)$

$Cb_{max} \leftarrow Max(pnt^{(WO,1)}_3, pnt^{(WO,2)}_3, pnt^{(BO,1)}_3, pnt^{(BO,2)}_3)$

Return $L_{WO}, L_{BO}, Ca_{min}, Ca_{max}, Cb_{min}, Cb_{max}$

The theoretical values are adjusted by 10% to account for the auto brightness and contrast adjustment performed by the phone as described in Section 3.2.3. We only wish to adjust the bounds away from the luminosity axis. One of the chromatic bounds will be the luminosity axis $Ca = \frac{1}{2}$ or $Cb = \frac{1}{2}$, as this is the white or black point. Whether the bound is the upper or lower limit depends on the starting point, so we define a function which performs the adjustment appropriately.

$$slack(x) = \begin{cases} (1-a)x & x < \frac{1}{2} \\ (1+a)x & x > \frac{1}{2} \\ \frac{1}{2} & x = \frac{1}{2} \end{cases} \quad \text{where} \quad a = 0.1 \tag{D.1}$$

We can now write the condition for whether a value could have suffered from white-out or black-out.

$$\begin{aligned} ((0 \leq L \leq slack(L_{BO})) \vee (slack(L_{WO}) \leq L \leq 1)) \wedge \\ slack(Ca_{min}) \leq Ca \leq slack(Ca_{max}) \wedge \\ slack(Cb_{min}) \leq Cb \leq slack(Cb_{max}) \end{aligned} \tag{D.2}$$

D.1.2 The Region Classification and Partitioning Function

It is desirable to classify a pixel value after rotation but before redistribution in-order to avoid unnecessary work on pixel values which are not of interest. This can be done to a degree, but it is necessary to assume reasonably uniform lighting conditions or by using boundaries set by the WOBO algorithm described in Section D.1.1 above. The pixel values are classified by chromatic value as either inside the target region or outside in each chromatic axes, dividing the chromatic plane into nine regions, as illustrated in Figure D.1. A rudimentary partitioning is performed by assigning 8 constant chromatic vectors to the values outside the target region around the mean. The chromatic channel values are not evaluated further for values outside the target region. The luminosity is always evaluated, because all that is required is a scaling from *qRsRange* to the destination range *dRange* and the luminosity is needed to choose appropriate bounds using the WOBO algorithm.

D.1.3 Floating the Mean

Different lighting conditions can affect the detected pixel color. Not accounting for strongly-colored light, the affect of different lighting conditions is to increase or decrease the satura-

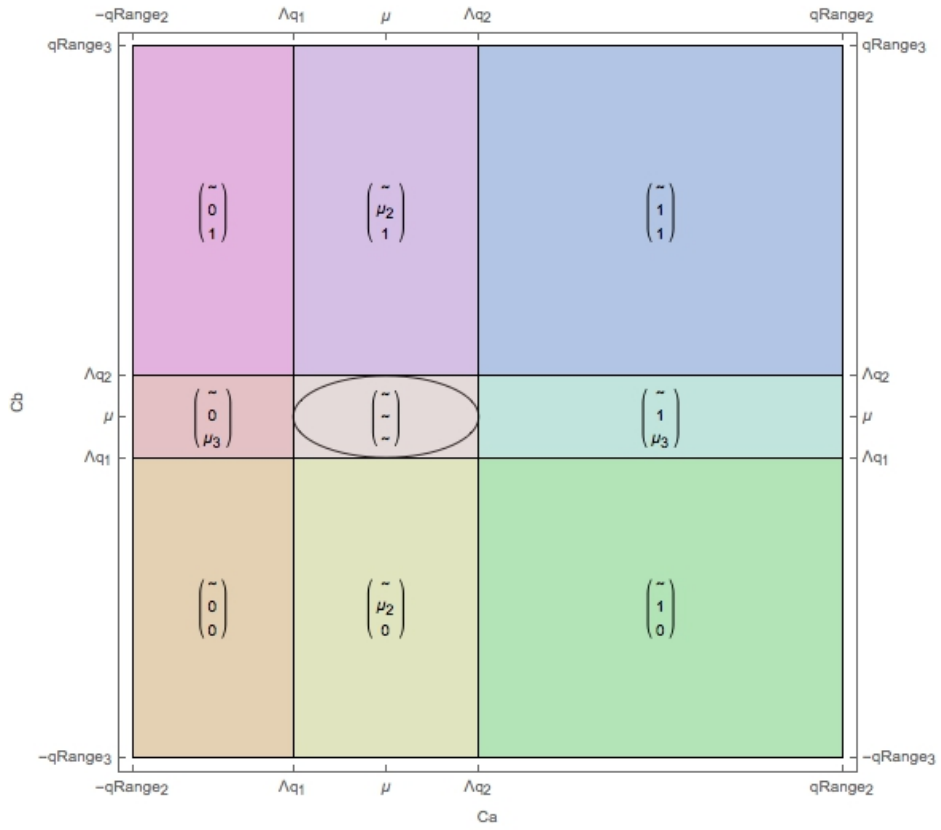


Fig. D.1 The Regions in the chromatic space. A $\sim$ indicates values which will undergo redistribution. Regions outside the target area are allocated extreme values at the edge of the color space

tion of the color. The distribution is relatively unaffected in orientation or variance, meaning that lighting conditions can be accounted for by adjusting the mean. The algorithm adjusts to ambient lighting by first taking a reference image centered on a patch of skin and finding the mean in the LCaCb color space. This new mean $\widetilde{\mu}$ is then used in the current instance of the algorithm. It is adventitious to keep the mean adjustable in the routine, so this is implemented by introducing an adjustment parameter $\delta\mu$ to the routine, where $\widetilde{\mu} = \mu + \delta\mu$. The algorithm assumes that the adjustment is small enough that the distribution function retains the same shape aside from a translation along the axis. This allows the distribution parameters to be adjusted by simply shifting them by $\delta\mu$.

$$\begin{aligned} \widetilde{\lambda}_1 &= \lambda_1 + \delta\mu \quad \widetilde{\lambda}_2 = \lambda_2 + \delta\mu \\ \widetilde{\omega}_1 &= \omega_1 + \delta\mu \quad \widetilde{\omega}_2 = \omega_2 + \delta\mu \end{aligned} \quad \text{Where} \quad \lambda_1 < \delta\mu < 1 - \lambda_2 \tag{D.3}$$

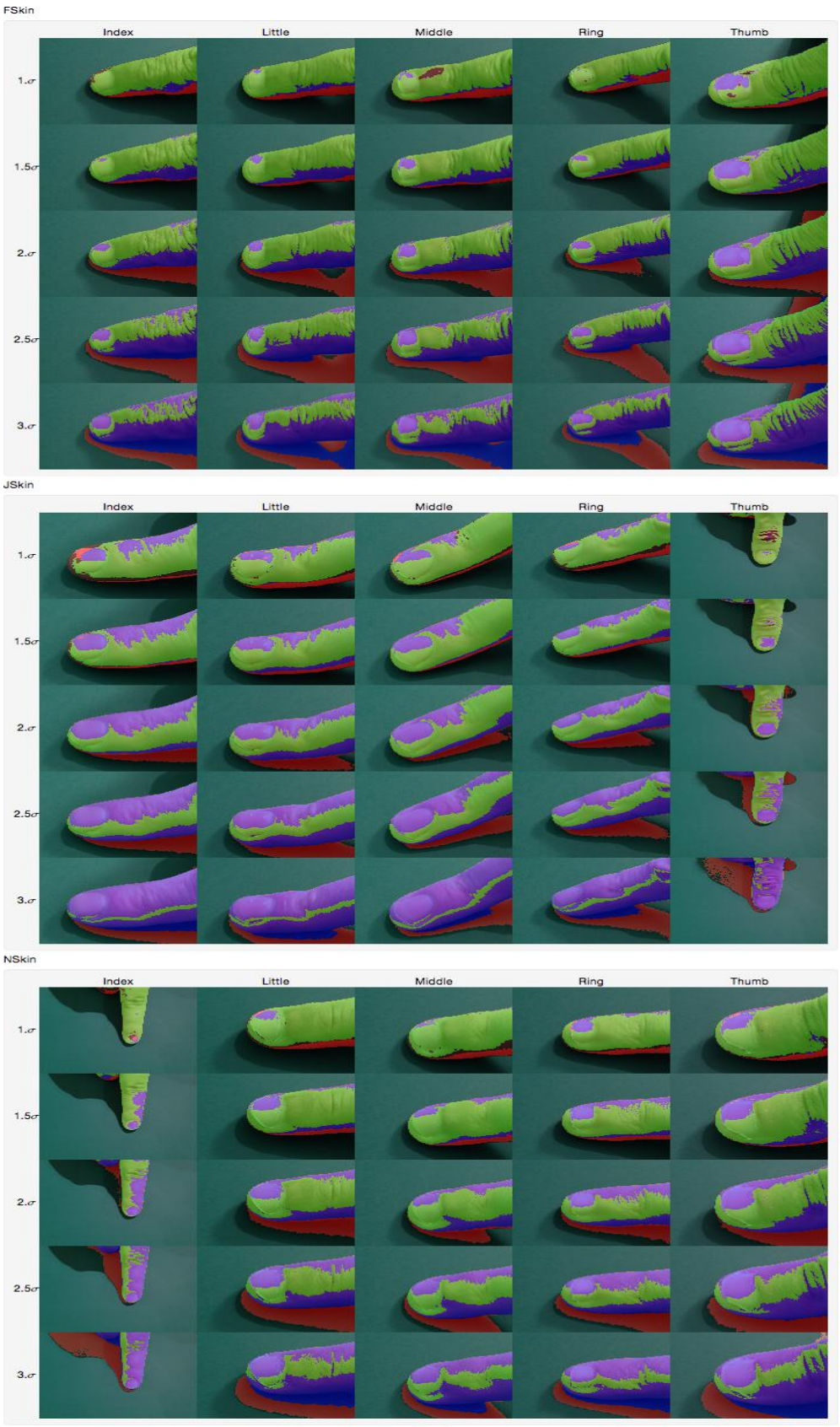


Fig. D.2 A selection of images classified using the quaternary method outlined in Section D.2.1, with various values of σ. It can be seen that 1σ correctly classifies the majority of pixels, however it classifies some on-digit pixels as "probably not skin", and even "not skin". 3σ, meanwhile, includes large portions of shadow as "probably skin". A compromise of 2σ-2.5σ is therefore suggested.

D.1.4 Loosened Deviation

The standard deviation found in Chapter 3 is a very tight fit to the chromatic values taken under controlled lighting conditions. For our practical implementation, this tight fit needs to be relaxed. This is done empirically by trying out multipliers of the standard deviation σ on the image sets. What is desired is a multiplier which relaxes the redistribution such that all the pixel values on a given digit are classified as skin, and all the background is classified as not skin. This is done empirically using the quaternary classification presented in Section D.2.1 below, which accounts for white-out and black-out and classifies the pixel value as "definitely skin", "probably skin", "probably not skin", and "not skin".

Using a value of 2.3 times the original σ correctly classifies the pixel values; this was found by observing tables of images similar to those in Figure D.2.

D.1.5 The Color Space Algorithm as Implemented

Here we present the color space algorithm in a way which most closely resembles the C++ code. The construction of the color space object is described at the end of Chapter 2; here, the operation of the algorithm is described.

First, we determine the form of the distribution function for a working type *tRange* with the source range $srcRange = 2^n$ and destination range $dstRange = 2^m$ expressed in terms of their bit depths.

$$K = 2^{m-n} \qquad \text{tRange}(\theta,\mu,\sigma) = \mathbf{l}\,2^n \qquad \mathbf{l} = \min\left\{2^{m-n}\delta(\mu,\sigma)\,,\mathbf{L}(\theta)\right\}$$

$$\text{tMin}(\theta,\mu,\sigma) = \begin{pmatrix} 0 \\ -\mathbf{l}_2\,2^{n-1} \\ -\mathbf{l}_3\,2^{n-1} \end{pmatrix} \qquad \text{tMax}(\theta,\mu,\sigma) = \begin{pmatrix} \mathbf{l}_1\,2^n \\ \mathbf{l}_2\,2^{n-1} \\ \mathbf{l}_3\,2^{n-1} \end{pmatrix} \qquad \vec{S} = \mathbf{l} \otimes \begin{pmatrix} \frac{1}{3} \\ 2^{1-n} \\ 2^{1-n} \end{pmatrix}$$

Substituting into the distribution function C.2 gives us

$$\mathbf{dis}(x) = -\frac{2^n\left(\text{erf}\left(\frac{2\mu+1}{2\sqrt{2}\sigma}\right)+\text{erf}\left(\frac{2^{1-2n}x-\mu}{\sqrt{2}\sigma}\right)\right)}{\text{erf}\left(\frac{2\mu-1}{2\sqrt{2}\sigma}\right)-\text{erf}\left(\frac{2\mu+1}{2\sqrt{2}\sigma}\right)} \quad \textbf{where} \quad \begin{aligned} qMin &< x < qMax \\ 0 &< \mu < 1 \\ 0 &< \sigma < 1 \end{aligned} \tag{D.4}$$

This form takes the result of the rotation directly, however we wish to add the flexibility to adjust the mean and use a lookup table for the values. It is desirable to keep the lookup

table as short as possible. The maximum compression of the information without losing relevant information is given by compressing to *tRange*. The number of values in the lookup table can be reduced by scaling x by a range *qtRange*, which is chosen to balance lookup table size and computational scaling efficiency. Noting that $\mathbf{l} < 2$, *qRsRange* can be chosen to be equal to *tRange* as if $\mathbf{l} = 2$, which means that *qtRange* is a power of two, allowing the scaling to be performed by bit shifting the value of $x_{qt} = x \ll n-2$. However, in the region where the function is applied, the information is preserved at best 2-to-1, so the information density can be reduced by half $x_{qt} = x \ll n-1$. Adjusting the mean changes the bounds on the region to be redistributed such that the form of the function does not change. For this reason, we also define x relative to the bounds, which allows the same lookup table to be used regardless of any adjustment to the mean.

$$\mathbf{dis}(x) = \begin{cases} \mathbf{disLU}((x-\Lambda_1)qtRange) & \Lambda_1 < x < \Omega_{pq1} \\ \mathbf{disLU}((x-\Omega_{pq2})qtRange) & \Omega_{pq2} < x < \Lambda_{q2} \\ \mathbf{dis}(x) & \text{Otherwise} \end{cases} \quad \text{where} \quad qtRange = \begin{pmatrix} \cdots \\ 2^{1-n} \\ 2^{1-n} \end{pmatrix} \tag{D.5}$$

Aside from these refinements, the algorithm presented in Chapter 2 is used to perform the rotation and redistribution as described. The algorithm proceeds as follows:

- Interactively find an adjustment to the mean $\delta\mu$.
- Set the new distribution parameters.
- Rotate the pixel value with the **qR** matrix: $pxl_q = \mathbf{qR} \cdot pxl_{RGB}$ then $\frac{-qRsRange}{2} < pxl_q < \frac{qRsRange}{2}$
- Partition the rotated values to avoid unnecessary processing of irrelevant information.
- Apply any user supplied per-pixel functions to the rotated value and add as extra channels.
- Redistribute the rotated values into the destination type.
- Apply and user supplied per-pixel functions to the redistributed values.

D.2 Pattern Recognition Routines

In this section, pattern recognition routines which use the new color space are presented and discussed.

D.2.1 Quaternary Pixel Classification

We wish to classify the pixel values as either skin or not skin whist taking account of the reliability of the detected color due to whiteout and blackout effects. This is done by applying equation D.2 and setting a boolean flag $\mathbb{L}_{\text{WoBo}}$ accordingly. The square partitioning performed earlier in the algorithm also sets a boolean flag $\mathbb{C}_{\text{square}}$ to 1 if the pixel is within the target range and to 0 otherwise. These can be combined to produce a 2-bit image which indicates both if the pixel value is in the target chromatic range and the reliability of the classification.

Correct	Unreliable	Image Bit		Comment
Color $\mathbb{C}_{\text{square}}$	$\mathbb{L}_{\text{WoBo}}$	$\mathbb{C}_{\text{square}}$	XOR($\mathbb{C}_{\text{square}}$, $\mathbb{L}_{\text{WoBo}}$)	
0	0	0	0	Definitely not the target
0	1	0	1	Possibly the target but shifted out of the target chromatic region by WOBO.
1	1	1	0	Probably the target but possibly shifted into the target chromatic region by WOBO
1	0	1	1	Definitely the target

This produces a 2-bit image, as seen in Figure D.3. For this purpose, a 2-bit data type was added to OpenCV. The question then is how to use it. Essentially, we have four different states: values which are neither the right color nor in the right region; values which are possibly the right color, but because they're outside the valid region, they've potentially suffered from white-out and black-out; values which are in the right region, but the wrong color, and we are certain that they are not skin; and finally, values which are in the valid region and are of the right color. In the following subsections we will present some useful methods which take advantage of this quaternary classified image.

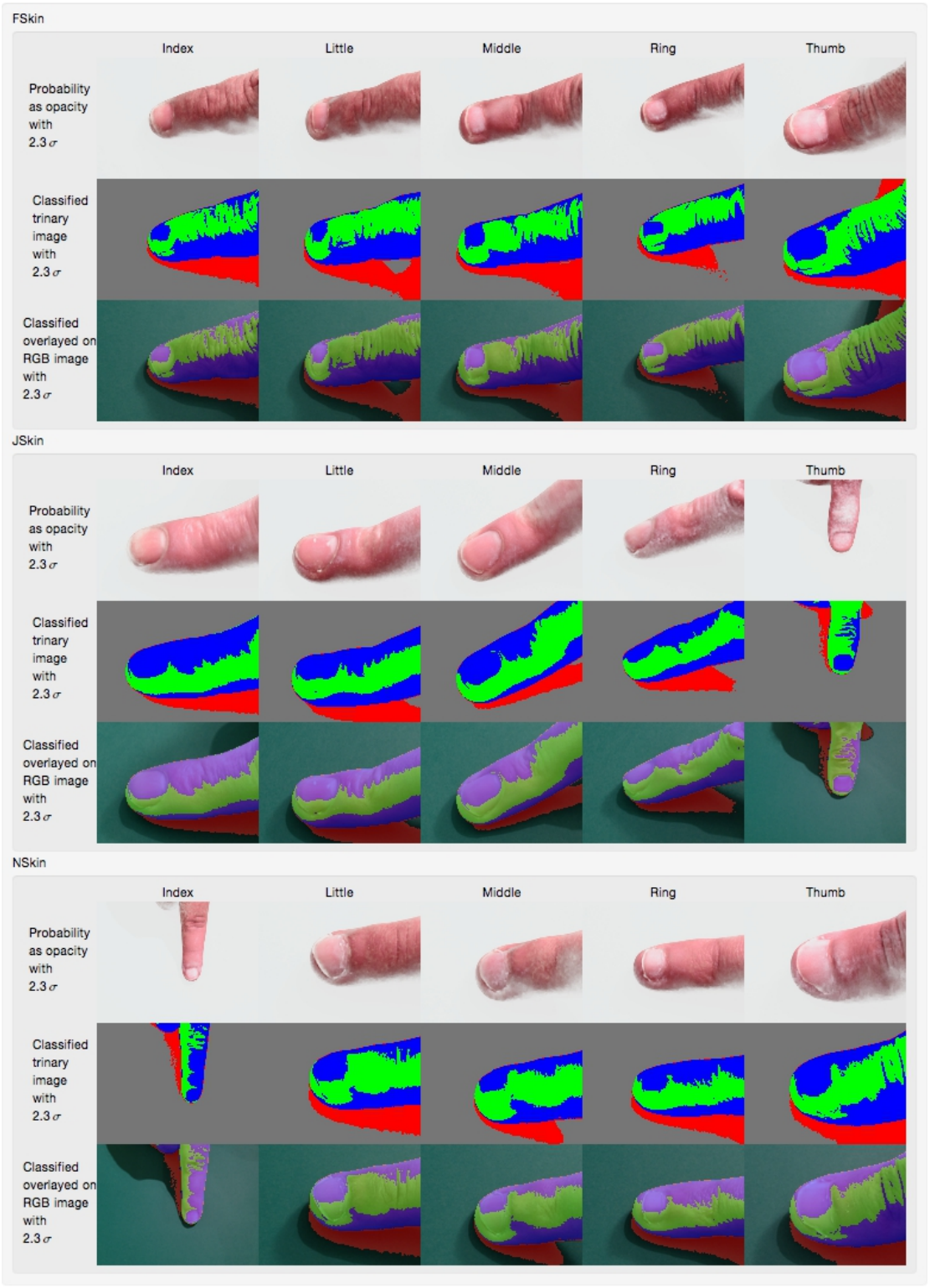


Fig. D.3 The Quaternary Pixel Classification presented alongside the probability scored image for comparison.

Appendix E

Chapter 4 Shape Finding Methods

E.0.1 The ‘Hurdle’ Method

The Hurdle method is an algorithm for finding a path inside an object using the quaternary classified image. The Hurdle method follows straight line paths in the object. As written, it is flexible enough to follow any straight line, however the current implementation takes a direction vector and evaluates at multiples of that direction vector. This means that non-horizontal and non-vertical lines are not guaranteed to be continuous. This is because the pixel values are not found using Bresenham’s line algorithm. The algorithm has two thresholds — a high and a low threshold. Whilst the pixel values remain in the high threshold, the path end point variable is updated; when the pixel values are in the lower threshold, the algorithm continues along the path, but without updating the path end point. This allows the path to pass through artifacts (i.e. pixel values which are inside the object but which have been misclassified), but does not allow the path to extend outside of the object into regions which are, for instance, in the shadow of the object.

E.0.2 The ‘Filament Fill’ Method

The Filament Fill method takes a path — given as a sequence of points — within an object and finds the edges of the object using the Hurdle method running away from the path within the object. Assuming a horizontally-oriented object, the path’s extent in the horizontal direction is divided equally into points, and these points are used to start the Hurdle method in the vertical directions, and vice versa for vertical-oriented objects.

For the purposes of the shape detection algorithms, there are metrics that are applied which suffer from noise and anomalies, where the noise is due to the natural variability of human digits from regular geometric forms, and the anomalies are due to misdetections of the edges of the digit. We assume that the anomalous results are extreme values, however we acknowledge that the most common (mode) or middle (median) values may not be representative of the best geometric approximation for the shape. The Mean-Median method — similar to the "k-Nearest Neighbors" algorithm — solves this problem by taking a confidence interval around the median and then taking the mean of the points within this interval. This method is a useful adaptation of methods like the k-Nearest Neighbors in that it solves the outlier problem for this particular application.

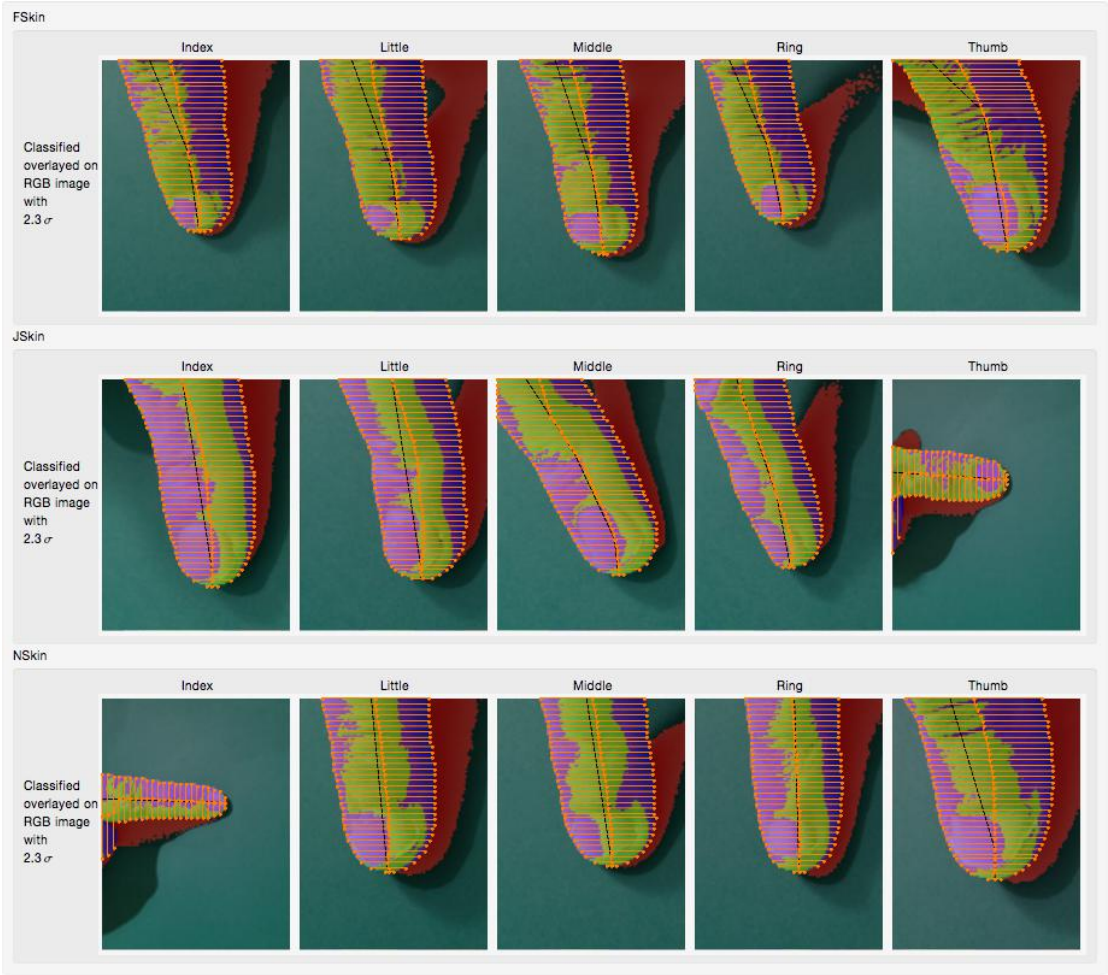


Fig. E.1 The 'Filament Fill' Method.

E.0.3 The 'Kink Fit' Method

The Kink Fit algorithm takes in a set of 2D points, where the first component is taken to be the independent variable and the second component is taken to be the dependent variable. This assumption excludes the possibility of truly vertical lines being presented to the algorithm. The Kink Fit algorithm fits a function which consists of two straight-line segments to a set of points. Unfortunately, currently the only piecewise linear fitting function is the proprietary MARS algorithm which, aside from being commercial, is overkill for this particular problem, which has a small number of points which are well-aligned.

The Kink Fit algorithm relies on the fact that we know that the points are in order; it chooses a point and divides the set in two, and then performs a standard, least squares linear fit to the two sets. Both sets include the point at which the set is divided.

The fit is also assumed to be continuous, except in the first derivative. The algorithm as

described above may produce two lines which do not intersect between the second-to-last data point used for the first line and the second data point used for the second line. The algorithm needs to ensure that this is the case. This is achieved by shifting the second half of the data set by the end point of the linear fit to the first half of the data set. The point at which the two linear fits intersect is found, and if that point is between the second-to-last point in the first set, and a second point in the second set (i.e. the points on either side of the point of division), then the fit is considered to be acceptable. If the fit is not acceptable, it indicates that there's a discontinuity in position rather than simply being a discontinuity in the first derivative. This is handled by choosing the point on the fit to the first set closest to the intersection within the bounds of the points on either side of the point of division. The linear fit is then performed with only the gradient as the free parameter, guaranteeing that the line will have passed through this limiting point.

The algorithm then finds the residuals, and then finds the average of the square of the residuals. This is considered the score for dividing the set of points at that position. The algorithm finds the point which minimizes the score using a bisecting algorithm; if a linear fit to the data has a low score, then the Kink Fit algorithm is considered unnecessary and the digit is assumed to be in a relaxed, straight, neutral pose. The algorithm specifies a minimum number of points to be included in a set, as this avoids the problem of dividing into sets which contain only one point.

E.0.4 The 'Elliptical Fit' Method

Fitting an ellipse to a set of points is surprisingly difficult. The reason for this is outlined below.

To follow the standard linear algebra approach to least squares fitting, we need to decide upon a functional form for the fit, and thereby a basis set. For an ellipse, this basis set is $\chi = \left(x^2, xy, y^2, x, y, 1\right)$, which is a set of six free coefficients $A^T = \left\{A_{\mathrm{xx}}, A_{\mathrm{xy}}, A_{\mathrm{yy}}, A_x, A_y, A_0\right\}$. However, to specify an ellipse, all that is needed is five numbers; the major and minor axes lengths (a,b), the position (x_0, y_0), and the orientation θ. This is because the basis set includes lines, quadratics, parabolic and hyperbolic functions as well as elliptical functions as possible fits. An additional difficulty arises when we try to construct a cost function for an ellipse. Ideally, we wish to minimize the perpendicular distances of the points from the ellipse, but the calculation of the distance of a point from an ellipse is not straightforward, requiring the roots of a fourth degree polynomial to be found. Due to these complexities, several different methods have been proposed by different authors, but we will be focusing

on two methods: The Approximate Mean Square (AMS) proposed by (**?**), and the Direct least square (Direct) method by (**?**).

For the fingertip problem, there is also the added difficulty that the points are confined to one side of the ellipse so we have chosen methods which cope well with partially occluded ellipses. Both methods begin by forming the design matrix and then applying an additional constraint which restricts the fit to a family of curves. Both methods are formulated as generalized eigenvalue problems, avoiding the need for iterative methods.

The AMS method restricts the fit to parabolic, hyperbolic and elliptical curves by imposing the condition that $A^T(D_x^T D_x + D_y^T D_y)A = 1$ where the matrices Dx and Dy are the partial derivatives of the design matrix D with respect to x and y. The matrices are formed row by row applying the following to each of the points in the set:

$$D(i,:) = \left\{x_i^2, x_i y_i, y_i^2, x_i, y_i, 1\right\} \quad D_x(i,:) = \{2x_i, y_i, 0, 1, 0, 0\} \quad D_y(i,:) = \{0, x_i, 2y_i, 0, 1, 0\}$$

The AMS method minimizes the cost function

$$\varepsilon^2 = \frac{A^T D^T D A}{A^T (D_x^T D_x + D_y^T D_y) A^T}$$

The minimum cost is found by solving the generalized eigenvalue problem.

$$D^T D A = \lambda \left(D_x^T D_x + D_y^T D_y\right) A$$

The Direct method confines the fit to ellipses by ensuring that $4A_{xx}A_{yy} - A_{xy}^2 > 0$. The condition imposed is that $4A_{xx}A_{yy} - A_{xy}^2 = 1$ which satisfies the inequality and as the coefficients can be arbitrarily scaled is not overly restrictive.

$$\varepsilon^2 = A^T D^T D A \quad \text{with} \quad A^T C A = 1 \quad \text{and} \quad C = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The minimum cost is found by solving the generalized eigenvalue problem.

$$D^T D A = \lambda \left(C\right) A$$

The system produces only one positive eigenvalue λ which is chosen as the solution

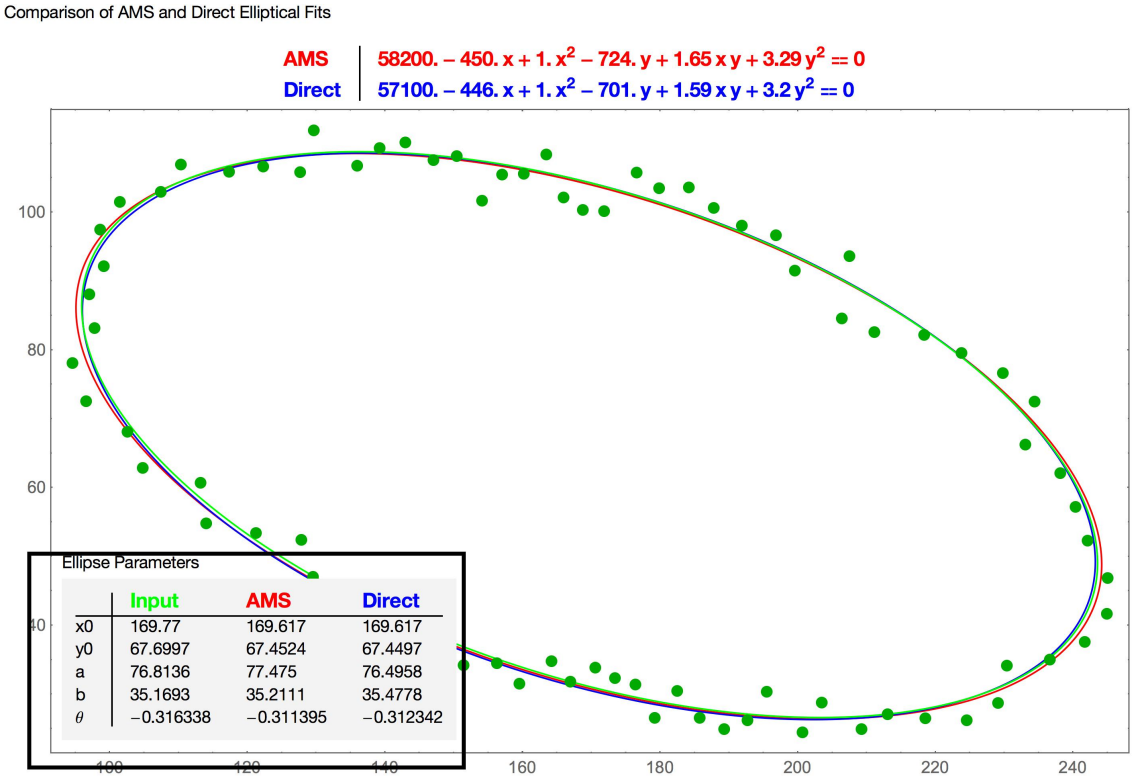
with its eigenvector **u**. These are used to find the coefficients

$$A = \sqrt{\frac{1}{\mathbf{u}^T C \mathbf{u}}}\mathbf{u}$$

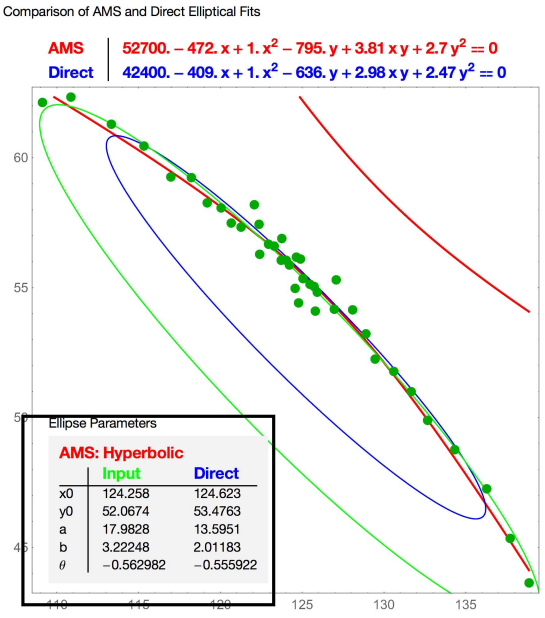
The scaling factor guarantees that $A^T CA = 1$.

These methods were implemented in Mathematica for evaluation and in the OpenCV C++ code. Testing routines generated sample data with variable ellipse occlusion and noise around a randomly generated ellipse . The methods both produced good fits to complete ellipses Fig.E.2a and incomplete ellipses Fig.E.2c. The AMS method was slightly better at fitting the extreme ends of the ellipse, generally producing a larger ellipse than the Direct method. Both methods accurately obtained the position and orientation. The AMS method occasionally produced a hyperbolic fit for very flat ellipses Fig E.2b.

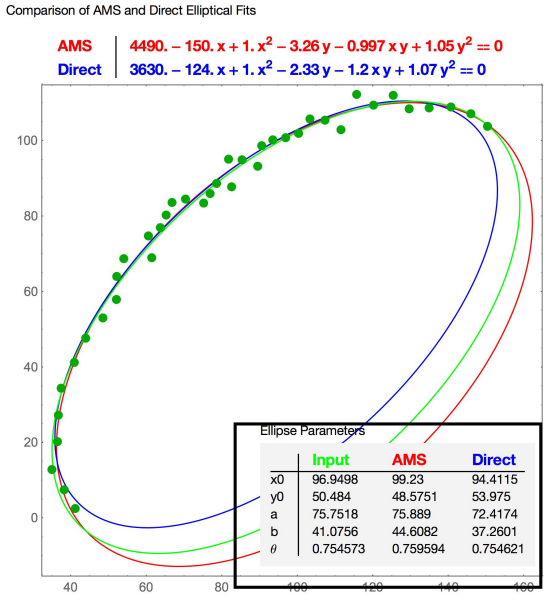
The largest computational effort in both methods is the determination of the $D^T D$ matrix. Once found, however, both methods can be used at relatively little extra computational cost. All Ellipses between the two can be considered as valid fits also. This suggests a method which allows for extra information to be used to refine the fit. In the case of the finger model this is the orientation, distal width and position from the parallel line fit. The ellipse between the two fits which most closely matches the extra information is chosen as the best fit.



(a) AMS and Direct methods both perform well with a set of points evenly distributed arround the ellipse.



(b) The AMS method sometimes produces a hyperbolic fit whilst the direct method always produces an ellipse



(c) The AMS method better fits the extreme edge points in the data set resulting in the AMS method producing larger ellipses than the Direct method. The true ellipse is most often between the two.

Fig. E.2 Comparison of the AMS and Direct ellipse fit methods.

E.0.5 The 'Isoscelian Trapezian' Fit

Given three sets of points corresponding to the top, middle and bottom of an isosceles trapezium which is oriented so that the parallel sides are on the left and right, a fit can be found as follows:

1. A straight line fit to the midpoints is found giving start and end points along with an orientation.
2. The points are translated and rotated so that the first middle point is at the origin and the last middle point is on the x axis.
3. The bottom points are reflected about the x axis.
4. A single straight line is fitted to the top and reflected bottom points together.
5. Four points describing the trapezium are found and returned along with the orientation and position in the original coordinates.

E.0.6 The 'Force Analogue Shape Detection' Method (FASDM)

The FASDM is one of the most straightforward — if not the most straightforward — shape alignment algorithms. Given a set of points, if we know a corresponding set of points on the the target shape model, we can find translation vectors between the points on the model $\mathbf{d}_i^{mdl}$ and the points on the image $\mathbf{d}_i^{img}$; the FASDM makes the analogue between these vectors and a force acting on that point on the model $\mathbf{F}_i = \mathbf{d}_i^{img} - \mathbf{d}_i^{mdl}$. The position and orientation of the shape is then found by translating and rotating the model such that the 'force' acting upon it sums to zero. To facilitate the calculation the model is expressed in coordinates where the origin corresponds to the center of mass for the model.

It should be noted that other transformations can easily be included, however for the current purposes, rotation and translation are all that is required. The translation vector $\mathbf{F}_i$ is found by taking the average of the force acting on the center of mass $\mathbf{P} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{F}_i$. Translating the model to this position allows the residual turning forces to be found $\mathbf{F}_i^{\tau} = \mathbf{F}_i - \mathbf{P}$. The moment about the center of mass τ_i is found simply by finding the magnitude of the outer product of the residual turning force $\mathbf{F}_i^{\tau}$ with the the point on the model $\mathbf{d}_i^{mdl}$ thus $\tau_i = \|\mathbf{F}_i^{\tau} \wedge \mathbf{d}_i^{mdl}\|$. The rotation which would reduce the moment to zero is found by taking the angle between $\mathbf{d}_i^{mdl}$ and $\mathbf{d}_i^{mdl} + \mathbf{F}_i^{\tau}$. The moment weighted average of these is the angle θ which will most reduce the total rotational moment τ.

$$\theta = \frac{1}{\tau}\sum_{i=1}^{n}\tau_i\theta_i \qquad \theta_i = \arccos\left(\frac{\mathbf{d}_i^{mdl}\cdot(\mathbf{d}_i^{mdl}+\mathbf{F}_i^{\tau})}{\|\mathbf{d}_i^{mdl}\|\|\mathbf{d}_i^{mdl}+\mathbf{F}_i^{\tau}\|}\right) \qquad \tau = \sum_{i=1}^{n}\tau_i$$

$$\mathbf{P} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{F}_i \qquad = \arccos\left(\frac{\mathbf{d}_i^{mdl}\cdot(\mathbf{d}_i^{img}-\mathbf{P})}{\|\mathbf{d}_i^{mdl}\|\|\mathbf{d}_i^{img}-\mathbf{P}\|}\right) \qquad \tau_i = \|(\mathbf{d}_i^{img}-\mathbf{P})\wedge\mathbf{d}_i^{mdl}\|$$

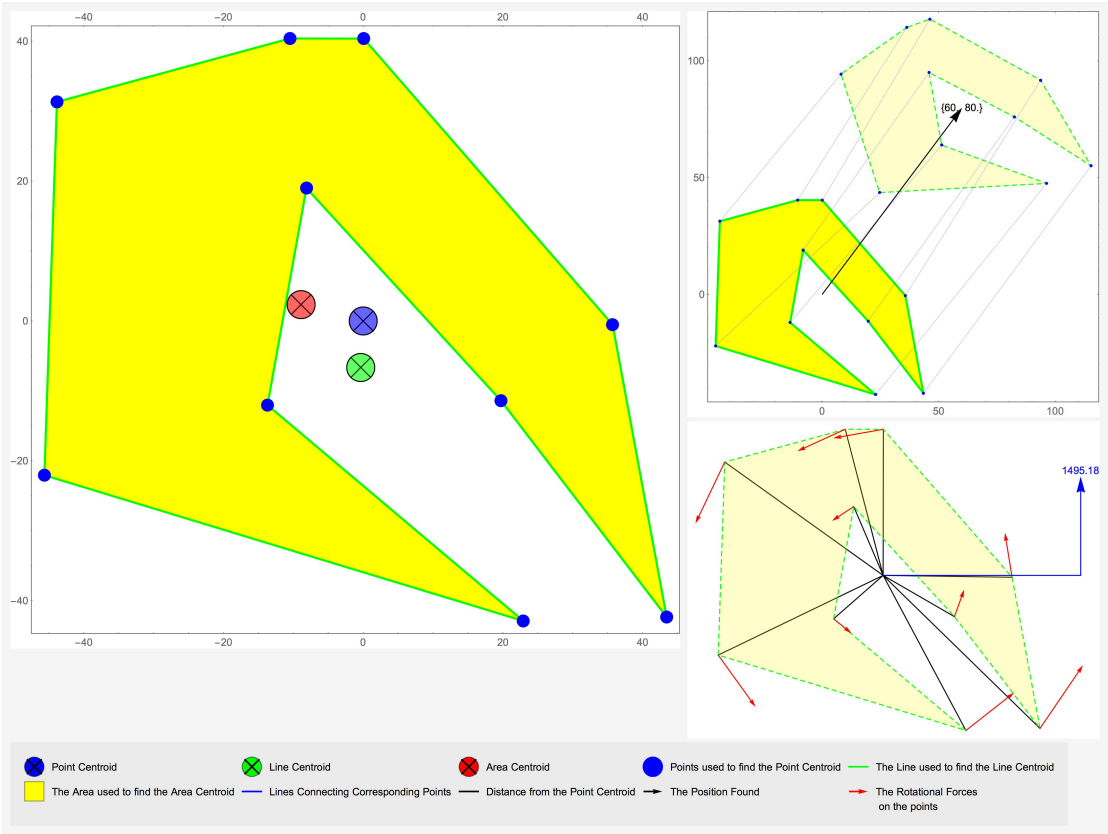


Fig. E.3 Random Polygon Centroids.

Mathematically and algorithmically, there are three points which may be considered the center of mass: the centroid of the area of the model, which is where the center of mass would be if the model were a a flat, uniform sheet; the centroid of the perimeter of the model, which is the center of mass would be if the model were constructed using wire along the perimeter; and the arithmetic average of the points of interest defined on the perimeter of the model. (See Figure E.3.) The choice for the analogue of the center of mass depends upon the algorithm used. For a method which returns points radially distributed about a point in the model would favor the area centroid; a method which returned points relatively evenly distributed along the perimeter would naturally favor the perimeter centroid method; and a

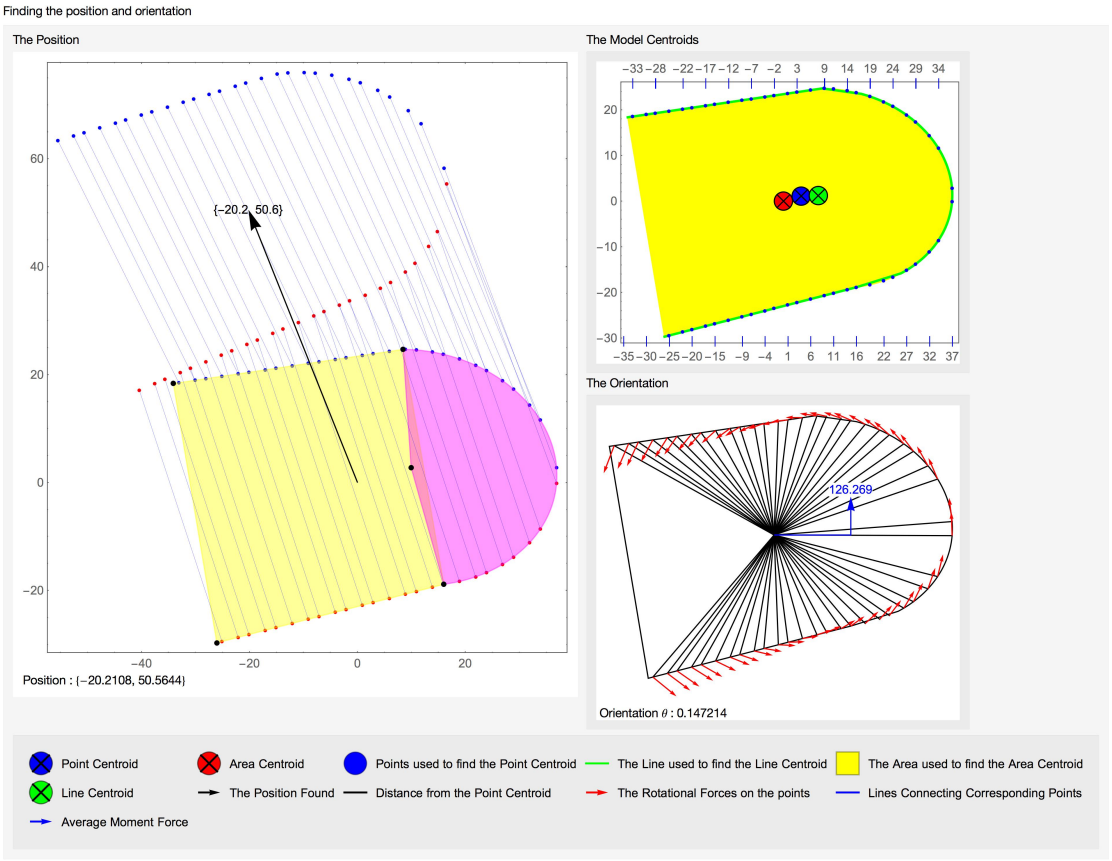


Fig. E.4 Position and Orientation.

method which returned points of interest would favor the point centroid method. (There's no guarantee the points of interest would be distributed in any relation to the shape-based centroids.) An example fingertip model centroid and FASDM method can be seen in Figure E.4.

The Filament Fill algorithm returns a set of points distributed linearly around the perimeter. The appropriate FASDM method is therefore a point-based centroid found by generating points along the shape perimeter which are equally-spaced perpendicular projections from the shape axis.