PRIFYSGOL BANGOR UNIVERSITY

Color Calibrated High Dynamic Ranged Timelapse Video using Remote Capture

Author: Delvin Varghese

School of Computer Science, Bangor University
May, 2014

Supervisor: Rafal Mantiuk

Acknowledgements

I would like to thank my supervisor Dr. Rafal Mantiuk for all his help and persevering with me, especially in the initial stages, when my knowledge in the field was not up to scratch. I would also like to thank my family, friends and peers for being there for me.

Statement of originality & release

Statement of Originality

The work presented in this dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Delvin Varghese

Statement of Availability

I hereby acknowledge the availability of any part of this dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein.

Student:

Delvin Varghese

Company Representative (if applicable)

.....

Abstract

High Dynamic Range images are known for their superior reproduction of real-world scene on a digital display. Traditionally, time-lapse photography has been done via standard low dynamic range images. This dissertation looks at ways to make HDR timelapses more feasible. It looks at making the whole process automated, as there is two-fold complexity introduced when combining HDR imaging with time-lapse photography. Not only does the user have to take multiple images (and manually change exposure time for each exposure) for each frame in the timelapse, but the user has to merge these exposures to form the tone-mapped HDR image, before using this in the timelapse. The program discussed here streamlines this process so that the user can set the parameters of the timelapse using an Android phone, and the system will generate an HDR timelapse , once completed. The camera used to capture the photos for the timelapse is colorimetrically calibrated against values measured with a spectroradiometer beforehand. This enables more accurate colour representation of the captured scenes in the timelapse video. Results of colorimetric calibration were published in a conference [1]. more details can be found in Chapter 4.2.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Timelapse videos are used to show slow changes in a scene within a short period of time. Scenes involving a lot of contrast benefit from using High Dynamic Ranged images. For example, photos taken with high amount of lighting e.g. harsh sunlight, bright incandescent lamp in scene, can cause glare, dark shadows and alter the metering on the camera, resulting in photos which are not desirable. HDR images take a weighted average of many photos, taken at different shutterspeeds, which helps even out many of the above mentioned problems.

A common problem that is ignored is the device-dependant nature of taking photos. Often, a photo taken with one camera appears to show different colours when compared to a photo of the same scene taken with a different camera. The same photo will show different colours when viewed on different display devices.

This dissertation proposes a system that automates the process of making a timelapse video made using colour calibrated HDR images. Colorimetric calibration is the process used to produce device-independent images. The camera is connected to a single-board portable computer (Raspberry Pi) and is controlled through this computer. A remote control interface is used using an Android phone, to set timelapse capture properties and to start the photography process remotely.

1.1 Objectives

- Colour calibrate camera to produce images with accurate colours.
- Capture photos using camera controlled by a portable computer. The computer is connected to the camera, and all photos are captured by commands sent by computer to camera.
- Initiate remote capture of photos from an Android Application interface. App enables user to set timelapse and HDR image creation parameters.

- Streamlined interface for making HDR timelapse using captured photos. This is used to make HDR images and then calibrate them. A timelapse video file is then made using the HDR images.

Results of colorimetric calibration were published in a conference [1]. more details can be found in Chapter 4.2.

Chapter 2

Background

2.1 Timelapse Photography

Timelapse is a cinematography technique in which the frequency at which film frames are captured (i.e. the frame rate) is much lower than the frequency at which the frames sequence is played back [4]. Timelapse makes a motion movie footage using a stills camera, utilising only still shots. [5]. A camera is set up to take multiple photographs of a scene. These photos are then merged into a video file using multiple frames per second of video.

This is ideal for scenes which changes over time but the changes are very subtle or not so pronounced. With the advent of cheaply available digital cameras, timelapse photography has become ubiquitous. One of the many reasons why this photography technique is used is because it can seamlessly show a scene and the changes within it in a condensed period of time such that the time progress / change in illumination is more pronounced. Popular uses include fruit rotting, road traffic, capturing natural phenomenon like sunsets, blossoming of a flower etc. For example, the blossoming of a flower [6], which can take many hours, is photographed at regular intervals (few seconds apart) and then merged into a short animated sequence or video at multiple frames per seconds.

2.2 High Dynamic Range Photography

The images used to generate the final timelapse video are captured using High Dynamic Range photography. This section provides more details on this.

2.2.1 Luminance

Luminance is the amount of visible light emitted or reflected from an object in a given direction [7]. It is thus an indicator of how bright the surface emitting/reflecting the light is. It is measured in candela per square meter (cd/m_2). Typical luminance values for common light sources [8] is given below:

1. Solar disk at noon : $1.6 * 10^9$ cd/m^2

2. Solar disk at horizon : 6 * 10^5 cd/m^2
3. Frosted bulb 60 W : 1.2 * 10^5 cd/m^2
4. Average clear sky : 8 * 10^3 cd/m^2
5. Moon surface : 2.5 * 10^3 cd/m^2
6. Average cloudy sky : 2 * 10^3 cd/m^2
7. Darkest sky : 4 * 10^{-4} cd/m^2

2.2.2 Dynamic Range

Dynamic Range is the ratio between the maximum and minimum luminance values of a scene. For instance, a scene showing the interior of a room with a sunlit view outside the window will have a dynamic range of approximately 100,000:1. Human vision can handle a very wide dynamic range. The human eye can accommodate a dynamic range of approximately 10,000:1 in a single view [9].

However, when a scene is captured using a digital camera, the full dynamic range of the scene is not captured by the camera sensor. Digital cameras use an array of millions of small light cavities called "photosites" to record an image. When a camera's shutter button is pressed, the photosites are uncovered for the duration of the shutterspeed, and then assessed to see how many photos were captured by each photosite [10]. A "colour filter array" is present over each cavity which permits either Red, green or Blue colours of light. By using a process called "Bayer Demosaicing", the array is of primary colors is converted into an image which contains full colour information at each pixel.

The size of the photosites determines the cammera's dynamic range [11]. If a photosite receives more photos than it is capable of holding, it becomes saturated and loses the additional photons received beyond its capacity. In practice, even if a camera could capture a large dynamic range, usable dynamic range is limited by the precision of the conversion of photosites' data to digital format by the camera. The maximum contrast ratio (ration of brightest colour to darkest colour) that can be expected from a digital camera is or **4096:1** (despite a camera's theoretical maximum of 16 Bit precision, the camera does not reach this maximum and 12 bits is generally expected, thus $2^{12} = 4096$ [11].

High Dynamic Range photography is a technique which, as the name implies tries to represent a larger dynamic range than permissible by the camera. This is achieved by taking multiple photos of the same scene, each with a different exposure level [9].

HDR photography ensures that there are no under-exposed or over-exposed pixels. As seen in Figure 2.1: Image A, Image B and Image C are three different photos of the same scene, taken with different exposure values. Image A contains underexposed regions and Image C contains overexposed regions. The three images are then merged to make the HDR photo **Image D**.

2.2.3 Capturing multi-exposure HDR

For making a high dynamically ranged image, several low dynamically-ranged photos are taken at different shutterspeeds i.e. exposure-times. Initially, the ISO and the Aperture settings of the camera

Figure 2.1: High Dynamic Range photography example. (Source: "HDR image + 3 source pictures (Cerro Tronador, Argentina).jpg" `http://en.wikipedia.org/wiki/High-dynamic-range_imaging` by Mariano Szklanny is licensed under CC BY 2.0)

is fixed. Then, the exposure value of the camera is adjusted manually or by utilising a feature present in modern digital cameras called Auto Exposure Bracketing (AEB) [12]. Under AEB, when a photo is taken by the user the camera takes a photo with the current exposure setting and then it also takes a photo with lower and higher exposure settings i.e. a total of three photos with three different exposure values. The increment in exposure time used to choose the next exposure setting is determined by a setting called f-stop. The number of f-stops determines the "jump" from the current exposure to the next chosen exposure setting. The number of f-stops and number of exposures that can be used for AEB vary from camera to camera [13]. If AEB is not used, the exposures are taken by manually changing the shutterspeed values before taking each exposure. Each photo represents a

different section of the total dynamic range. The photos are then merged using software to make an HDR image. HDR images were merged using RAW photos.

2.2.4 RAW vs JPEG

A RAW file contains raw unprocessed information straight from the camera's sensor. The camera's CMOS sensor is very efficient at converting the photons that reach it during the exposure time to electrons and this charge is then stored digitally by the device. When a camera stores an image in JPEG format, it sacrifices a lot of information about the scene to achieve its trademark 8:1 compression ratio. A JPEG image undergoes various processes inside the camera: demosaicing, White Balance correction, Sharpening, Colour Saturation, Tone curves etc. [14]. A JPEG image is also a gamma corrected reproduction of the RAW data and it requires a response curve to retrieve a camera-independent linearly corrected image. A RAW format image on the other hand is linearly corrected. The JPEG image once produced contains several irreversible steps as the conversion makes it lose precision [15]. For example, JPEG files are also corrected for White Balance by the camera. To change this in post-production is not possible without sacrificing quality as the image is corrected twice (once in the camera, once in post-production) [16]. As discussed above, the camera can capture the information from the scene in 16 bits, giving precision. This is lost when compressing to JPEG which uses 8 bits per colour channel. JPEG format also exhibits compression artifacts (depending on the compression ratio) which the RAW format doesn't suffer from because of its low compression qualities [17]. Taking these factors into consideration, the exposures were photographed in RAW format.

2.3 Colorimetric calibration

Colorimetry is the science of measuring colours [18] and is widely used in commerce, industry and the laboratory to express colour in numerical terms and to measure colour differences between specimens [19].

Calibration is required to produce a device independent reproduction of a scene. Any device that captures or displays images should be calibrated so that the colours it shows are representative of the originally intended colour in the scene. The image in Figure 2.2 shows the same image on two different screens. Some of the patches exhibit noticeably different colours (esp. on Row 3 of the chart).

Similarly, different cameras taking photos of the same scene will have differences in colour when compared. Calibration is a corrective procedure applied to an image so the colours it uses to represent the scene are as close as possible to the real world colours.

2.3.1 What is Colour?

Often when discussing colour it cannot be described without resorting to examples. Colour is an attribute of visual sensation and the visual sensation of colour appearance of objects depends on the interaction between light sources, objects and the human visual system [20]. A light source is required

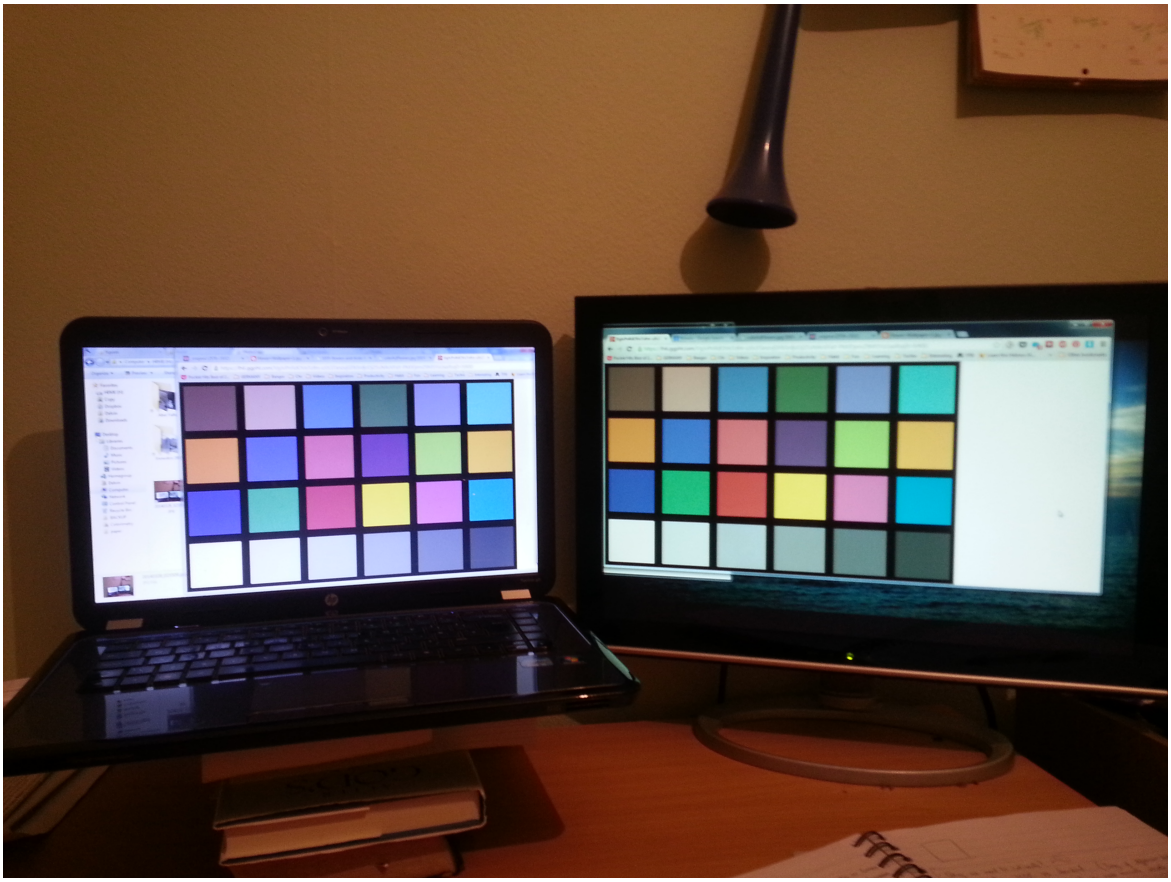


Figure 2.2: Two displays showing the same image.

since visible electromagnetic energy is necessary to initiate the sensory process of vision. When energy from the light source hits the object, the energy is either absorbed, reflected or transmitted according to the physical and chemical properties of the object.
The eye perceives the light energy, after it is modulated by the object's properties, by means of four kinds of cells on the retina: rods and long, medium and short cones. The cone cells' response to the energy forms the colour of a light as perceived by the eye. [21]

2.3.2 Colourspaces

For colorimetric calibration, the XYZ colourspace is used instead of the RGB colourspace traditionally associated with digital photography. RGB stands for Red Green Blue, and like the abbreviation implies, the RGB colourspace is an additive colour space defined by the **chromaticities** of the red, green and blue primaries [22]. Chromaticity refers to a two-dimensional description of colour, which corresponds to the combination of hue and purity, omitting the third dimension of brightness. The luminance (brightness) and chromaticity of a spot on a display, taken together, provide a complete description of its colour [23].

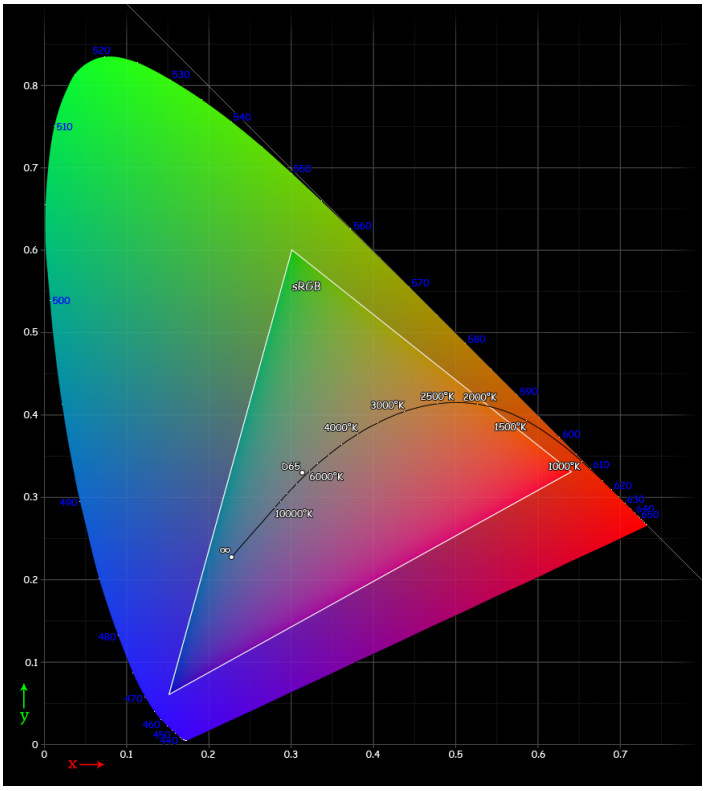


Figure 2.3: CIE 1931 xy chromaticity diagram showing the gamut of the sRGB colour space. This RGB colour space represents only a proportion of the colours that can be represented in the CIE XYZ colour space, shown by the horseshoe in the diagram. (Source: "Cie Chart with sRGB gamut"(`http://en.wikipedia.org/wiki/File:Cie_Chart_with_sRGB_gamut_by_spigget.png`) by Spigget is licensed under CC BY 2.0)

The RGB model is traditionally used in computer graphics and related applications because of its similarity to how the human visual system works [24]. However the range of colours that can be represented using the RGB model is limited. We describe this property as having a smaller **colour gamut** [25]. XYZ colour space was created to overcome this weakness and is based on RGB colour space itself. It has a much larger gamut and can represent all possible colour stimuli that the average human eye can observe.

The CIE XYZ 1931 colour space was one of the first mathematically determined colour spaces. It was derived by the CIE from the CIE RGB colourspace as it considered the red, green, blue tristimulus values to be undesirable for creating a standardized colour model. They used a mathematical formula to convert RGB data to a colour space that uses only positive integers as values [26].
The colour matching functions of the CIE 1931 standard colorimetric observer are used to calculate the XYZ tristimulus values (shown in equations 2.1, 2.2 and 2.3).

$$X = k \int_{\lambda} \phi(\lambda)\overline{x}(\lambda)d\lambda \tag{2.1}$$

$$Y = k \int_{\lambda} \phi(\lambda)\overline{y}(\lambda)d\lambda \tag{2.2}$$

$$Z = k \int_{\lambda} \phi(\lambda)\overline{z}(\lambda)d\lambda \tag{2.3}$$

where $\phi(\lambda)$ is the spectral power distribution of the stimulus (total luminance reflected or transmitted from the stimulus) and $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ and $\overline{z}(\lambda)$ are the colour matching functions, and k is a normalizing constant [20].

The standard observer is used to represent an average human's chromatic response within a 2 degree field of view, as when Wright & Guild conducted the experiment in 1931, it was believed that all the colour-sensing cones of the eye were located within a 2 degree arc of the fovea [27].

2.3.3 CIE xyY

In CIE xyY, Y is the luminance and x and y represents the chromaticity values derived from the tristimulus values X, Y and Z in the CIE XYZ colour space. The small x and y values are used to draw the chromaticity diagram of the CIE XYZ colour space. It is just another way to represent CIE XYZ. [28] Values in CIE xyY can be derived from XYZ tristimulus values [29] in this way:

$$x = \frac{X}{X + Y + Z} \tag{2.4}$$

$$y = \frac{Y}{X + Y + Z} \tag{2.5}$$

$$Y = Y \tag{2.6}$$

The CIE XYZ colour space was designed such that Y represented the luminance of a colour, hence it is unchanged when used in CIE xyY colour space.

2.3.4 Colour measurement

The camera is calibrated using a training scene. Initially the camera takes a photo of a scene with sufficient colours to provide samples for the calibration. Then a **spectroradiometer** or equivalent luminance measurement device is used to capture absolute luminance values (captured in Yxy colourspace but easily convertible to XYZ colourspace) of the colour samples.
A spectroradiometer is designed to measure spectral radiance or irradiance across various spectral ranges [30]. It has a built in optical measuring and targeting system measuring light from approximately 350nm to 1000nm. The spectral data can then be used to calculate CIE tristimulus values. The PC interface for the spectroradiometer captures the values in CIE Yxy colour space, which can be converted easily to CIE XYZ format as described in equations 2.7, 2.8 and 2.9. These values are

Figure 2.4: The specbos1211 spectroradiometer. (Image source: [2])

then used as the reference XYZ values for calibration purposes.

Figure 2.4 shows the spectroradiometer used to capture the luminance values.
The xyY values are converted into CIE XYZ space for the calibration process using the following equations [31]:

$$X = \frac{xY}{y} \tag{2.7}$$

$$Z = \frac{(1-x-y)Y}{y} \tag{2.8}$$

The Y luminance component is the same parameter in both colour spaces.

$$Y = Y \tag{2.9}$$

2.3.5 Measuring colour difference

Colour difference between two objects is usually expressed subjectively i.e. a variety of colours can each be called "green" or "red". When comparing two colours, traditionally one colour is said to have more "red" etc. The RGB and XYZ colour spaces were created by the CIE in 1931. However, these colour spaces are not perceptually uniform despite the accuracy of calculating the absolute differences i.e. a change of the same amount in two colour values doesnt produce the same noticeable difference. Some colours need to be altered by a higher value than others to produce similar change as observed by the eye.
Thus, when colour calibrating, a robust metric is required for measuring colour differences. Colour difference is denoted as δE (delta-E), a single number which denotes the distance between two colours. Colour differences are measured in the CIELAB colour space as the Euclidean distance between the coordinates for the two stimuli. For two sets of colour coordinates in CIELAB colour space (L*,

a* and b* denote the three dimensions of the coordinates in CIELAB), the colour difference can be represented as in Equation 2.10. This is the standard CIE colour difference method and known as **CIE 1976**.

$$\Delta E^*_{ab} = \left[\Delta L^{*2} + \Delta a^{*2} + \Delta b^{*2}\right]^{1/2} \tag{2.10}$$

CIE 1976 was designed with the goal of having colour differences be perceptually uniform throughout the space (i.e. a ΔE^*_{ab} of 1.0 for a pair of red stimuli is perceived to be equal in magnitude to a ΔE^*_{ab} of 1.0 for a pair of gray stimuli) [20]. However, this goal was not achieved **??**.

It was soon realised by colour researchers that CIE 1976 (ΔE^*_{ab}) was an inadequate metric, and CIE 1994 (represented as ΔE^*_{94}) was devised to overcome its deficiencies **??**. CIE 1994 is determined in the L*C*h* colour space (representing **L**ightness, **C**hroma, **h**ue of the stimuli). The colour difference according to CIE 1994 is shown in Equation 2.11 [20] [32]:

$$\Delta E^*_{94} = \left[\left(\frac{\Delta L^*}{k_L S_L}\right)^2 + \left(\frac{\Delta C^*_{ab}}{k_C S_C}\right)^2 + \left(\frac{\Delta H^*_{ab}}{k_H S_H}\right)^2\right]^{1/2} \tag{2.11}$$

where C^* and H^* are calculated from CIELAB coordinates **??**:

$$C^* = \sqrt{a^{*^2} + b^{*^2}} \tag{2.12}$$

$$\Delta H^*_{ab} = \sqrt{\Delta E^{*^2}_{ab} - \Delta L^{*^2} - \Delta C^{*^2}_{ab}} = \sqrt{\Delta a^{*^2} + \Delta b^{*^2} - \Delta C^{*^2}_{ab}} \tag{2.13}$$

S_L, S_C and S_H are given as:

$$S_L = 1 \tag{2.14}$$

$$S_C = 1 + 0.045 C^*_{ab} \tag{2.15}$$

$$S_H = 1 + 0.015 C^*_{ab} \tag{2.16}$$

k_L, k_C and k_H are set to 1 by default but are used to adjust the relative weighting of the lightness, chroma and hue components respectively according to viewing conditions and applications that differ from CIE94 reference conditions. CIE94 is the most widely used colour difference formula **??**.

In 2000, CIE adopted a new colour difference standard called CIE 2000 (denoted as ΔE^*_{00}) which adds corrections to CIE 1994 at the cost of added complexity to CIE 1994. Unlike CIE 1994 which assumes that L* crrectly reflects the perceived differences in lightness, CIE 2000 varies the weighting of L* depending on where in the lightness range the colour falls [33]. Colour difference according to the CIE 2000 equation is [34]:

$$\Delta E^*_{00} = \left[\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \frac{\Delta C'}{k_C S_C}\frac{\Delta H'}{k_H S_H}\right]^{1/2} \tag{2.17}$$

CIE 2000 introduces R_T, hue rotation parameter, for the blue-violet region in the colour space where the CIE 1994 formula lacked accuracy [35]. It also compensates for lightness(S_L), chroma(S_C) and

hue(S_H).
CIE 2000 substitutes the C, L, h values in CIE 1994 for C', L', h'. The conversion process for this is beyond the scope of this thesis, but it has been discussed in detail by Sharma et.al [36].
CIE 2000 is the most accurate colour difference metric, especially for small colour differences. Furthermore, the complexities introduced in this equation compared to CIE 1994 are handled efficiently in software. **Thus, CIE 2000 is the chosen metric for finding colour difference during the colorimetric calibration process**.

Chapter 3

Related Work

Section 3.1 details previous work in the area of colour calibration. A number of papers are examined and findings relevant to this dissertation are discussed.

Section 3.2 describes applications and systems that deal with High Dynamic Range timelapse videos. Some Android applications that provide related functionality are also discussed.

3.1 Colour Calibration

Collor calibration has previously been researched by a number of authors. This section briefly reviews a selection of related research which are relevant to the colorimetric calibration undertaken in this dissertation.

Inanici [37] tried to evaluate the applicability of using HDR photography as a luminance acquisition system. To take the exposures required for the HDR image, they used a Nikon Coolpix 5400 digital camera and fitted it with a fisheye(ultra wide-angle) lens. They also captured the luminance in the scene using a calibrated luminance meter (Minolta LS110). Using *Photosphere* software, they then generated the camera's response curve. Using the response curve, the photographs were fused together and stored in RGBE (an HDR image format).

To check for accuracy, the laboratory was illuminated with different light sources including incandescent and fluorescent lighting. The targets checked for luminance were greyscale and colour paper targets. The greyscale targets tended to have a lower error percentages than their colour equivalents. Compared with physical luminance measurements, they reported reasonable accuracy. The average error percentage for all targets was 7.3%, for greyscale targets it was 5.8%, and for colour targets it was 9.3%.

They concluded that HDR photography is not a direct replacement for luminance meter measurements, as the former needs calibration against a reliable reference source. However, they admitted that it has the advantage of being cheap and can efficiently collect luminance data within a large field of view quickly and efficiently, which is not possible with a luminance meter.

Similarly, Anaokar and Moeck [38] gauged the accuracy of using HDR Photography for luminance measurements. Like Inanici, they used the Nikon Coolpix 5400 and used *Photosphere* as the software of choice for HDR merging. They used six Munsell cards consisting of 14 chips each varying in Munsell values (brightness) and chroma (saturation).

They demonstrated that the error in reflectance increases as the Munsell value decreases. They also showed that the error in reflectance for all Munsell colours, irrespective of hue, chroma and saturation, is independent of the illuminance level on the target and that the error in reflectance was independent of the lighting condition.

The authors concluded that HDR photography can be used in scenes with materials having low chroma, saturation and warmer hues like building interiors and built environments. However, in scenes like foliage, which have cooler hues like blue and green having high chroma, the errors can be as high as 80%.

Moeck [39] engaged the topic of HDR Imaging once again, but this time the author decided to use a recently released CMOS sensor based camera(Canon EOS 350D, an 8MP DSLR camera), as opposed to the CCD cameras used in previous studies. They photographed Munsell grey cards, Munsell colour samples and matte colour checkers, with known reflectance and uniform illuminance under a clear sky and stored the exposures as JPEG images. Once merged in *Photosphere*, they calculated the error in luminance and found that dark colour samples with low Munsell value are overestimated and light colour samples are underestimated. They found blue(53%), purple blue(48%) and blue green(36%) to have the largest error while Yellow red, yellow and yellow green display the least error.

Moeck attributed his reasons for photographing in JPEG rather than RAW to disk space constraints. JPEG had the advantage of an 8:1 compression ratio, which was imperative to the study. He observed that the camera's RAW file format should be used for colorimetric applications similar to his, where file size is not an issue. This dissertation will be addressing this issue as RAW images will be used for all calibration purposes. The author compared the results of using a CMOS sensor camera in this study with the preceding study mentioned above, of which he was the co-author. In that preceding study, a CCD sensor camera was used. The maximum measured error for the Nikon with the CCD sensor was 32.85%, while the maximum error for the Canon with a CMS sensor was 16.89%.

Krawczyk, Goesele and Seidel [40] presented results from the photometric calibration of three camera systems (2 HDR and 1 LDR) to capture real-world luminance values. The HDR video cameras used were: Silicon Vision Lars III and HDRC VGAx. A Jenoptik C14 was chosen as the LDR camera used for comparison purposes.

Since the authors intended to cover a dynamic range spanning at least six orders of magnitude, they chose three target setups with varied luminance characteristics. The response curve for the three cameras was then recovered. However, to perform an absolute calibration, they used a Gretag-Macbeth ColorChecker chart and used a Minolta LS-100 luminance meter to determine the optimal scale factor for each camera. They proceeded to then compare the luminance values of the calibrated camera and they proved to be well aligned with the measured values proving that the response curve recovery was accurate. When they measured the relative error for the three cameras, the C14 LDR

camera proved to be most accurate.

The authors also tried to fit the recovered response curves of the cameras directly to the measured luminance values from the experiment using a pre-written function for each HDR sensor. They discovered that for high luminance values, the calibration via function fitting provided more accurate results. They attributed the low precision of results in the lower luminance range to either an excessive amount of noise or glare introduced by the ND filters used. The experiments showed the lack of accuracy of HDR cameras in dark conditions (below 10 cd/m_2)

In his research, Tyukhova [41] investigated whether HDR photography can accurately capture luminance emitting from a LED chip. They used a Canon EOS 7D and Canon EOS 500D, and a Neutral Density filter was used on the lens. The first step in their HDRi experiment was to extract the camera response curves in an interior office setting. Luminance values were obtained using the Minolta LS110 luminance meter. Next, the HDR exposures were taken and merged in *Photosphere*. In their extensive dissertation, they considered the two traditional ways of measuring luminance: using a luminance meter and direct illuminance measurements in a constructed photosphere.

They found that inconsistent accuracy between the two approaches could be credited to the small size of the LED, the assumption of treating a LED chip as a uniform light source and aiming problems. They found that for exposures taken at the same aperture size, HDRs merged from RAW images exhibited much less vignetting effect than the ones merged from JPEG images (15.3% and 20.0% respectively). HDR Imaging's ability to capture incandescent light sources compared to that of a luminance meter measurement was within 1.4% error at 28mm and 105mm focal lengths, and they expressed confidence in using RAW images for HDR photography and calibration. By contrast, due to the inconsistent nature of the luminance meter measurements, the error rate was about 35% for the same set.

Goesele et. al [42] approached the colour calibration problem from a different angle; they used ICC profiles to calibrate the low dynamic range images that were used to create the HDR image. ICC profiles describe the colour characteristics of image display and capture devices like cameras, monitors etc. and are used to provide a mapping from the colour space of the input device to a target colour space which may be device independent or for another device. They considered the use of a simple white balance between the red, green and blue channels of an image for colour calibration too simplistic. They acquired colour corrected images in XYZ space using an ICC input profile, which ensured a linear relationship between every pixel in the colour corrected image and its equivalent in a similar image with a different exposure setting. They used this property to scale and average the pixel values for the image set and created an HDR image. The authors noted that using an input profile is better than using a response curve as it not only linearises data but also converts in into the well defined CIEZYX colour space.

For their experiments, they used a Kodak DCS 560 camera to take 13 exposures of an ANSI IT8 target. They converted the captured exposures into 16 bits per channel XYZ colour space. They then calculated the response curves for all colour channels. Analysing the response curves confirmed that the resulting colour space is approximately linear. However, they discovered some problems like:

a bias in the response curves in the X and Y channels, signal is strongly non-linear and noisy in very dark regions, response curve has a high variance in very bright regions. To counteract these problems. they used a weighting function to suppress values at both ends of the dynamic range. They warned that the weighting function required might vary from setup to setup. According to their results, 50% of pixels in the image had an absolute error difference lower than 2.They concluded that once the linearity of the image in XYZ space is established, colour calibration using ICC profiles is easy to implement and that the quality of the results depends mainly on the input profile used and the bits per channel used on the input images' channels.

3.2 Remote Controlled HDR Timelapse

Partly due to the niche nature of a Remote-Controlled HDR Timelapse, not a lot of such projects exist. However, I have encountered a lot of interest in aspects of my dissertation which have been implemented by various enthusiasts. Most of the relevant projects fall in the following categories:

I. **Android Apps that control DSLRs by cable** [43, 44]: This involves an Android phone connected to a DSLR camera, where the Android acts as the USB host. This has the benefit that it is a compact and hassle-free setup with no other components required. Some of the apps in this category like **DSLR controller** [43] can set camera properties required for an HDR image such as exposure. If the camera supports Auto Exposure Bracketing, then **DSLR Controller** can set it up too, making the process of capturing an HDR easier. However, no facilities for creating a timelapse with the HDR captures exists.

II. **Regular Camera capture using Raspberry Pi** [45]: This is a very simple setup and can be easily implemented either using PiCam (a Camera module which can be attached to the Raspberry Pi) or by connecting a camera via USB cable and using gPhoto command line. This approach implements neither HDR photography or timelapse creation.

III. **HDR photography using Raspberry Pi**: Khurt Williams [46] designed a setup using a Canon D40 connected to a Raspberry Pi. The Pi would wait for a camera to be turned on and then would take 5 exposures by adjusting the EV exposure compensation setting on the camera using gPhoto running on the Pi. Once the exposures have been obtained, they are merged by hand later to form an HDR image.

IV. **Timelapse using Raspberry Pi** [47–51]: A number of good projects are on the internet which detail Timelapse capture using a Raspberry Pi. Marzolla [47] implemented the entire process by hand, i.e. he captured the exposures, then labelled them and then using an encoder, made a timelapse movie at 12 fps.

As can be seen from the nature of the applications above, they do not attempt to tackle all three branches of the problem I am trying to address:

1. Needs to be Remote Controlled
2. High Dynamic Ranged Images have to be used

3. Generate a Timelapse

Having a wide variety of such projects is beneficial however, as their implementation has provided inspiration in my own approach. All four approaches above have engaged different aspects of this dissertation, and as such my implementation will differ due to the complexity introduced by combining them.

Chapter 4

Calibrated HDR-Capture

4.1 Overview

The implementation of the thesis is divided into a number of distinct sections which are part of the unified complex system. The sections can be listed as:

1. Finding the calibration matrix by Colorimetric Calibraton.(Section 4.2)
2. Setting timelapse parameters via Android App to server application.(Section 4.3)
3. Server application controls the camera and takes exposures based on parameters received.(Section 4.4)
4. Merging to Calibrated HDR images and creating Timelapse from them.(Section 4.5)

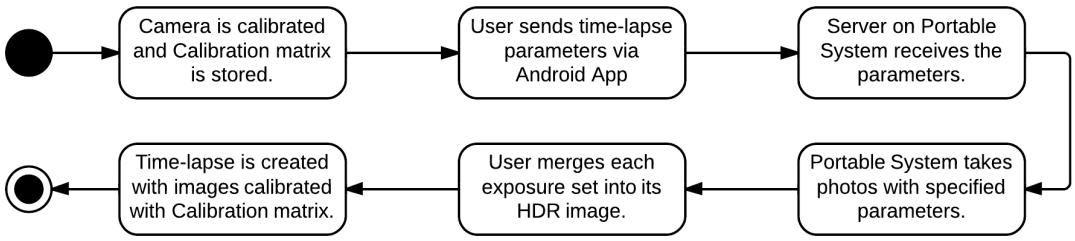


Figure 4.1: Activity Diagram of the Timelapse Acquisition System.

We find the calibration matrix required to calibrate the images in Section 4.2. In this step, we find the transformation matrix M, which will be necessary in Step 4. Step 2 describes the Android application and its setup. The app allows the user to set capture parameters like ISO, Aperture, Shutterspeed settings and timelapse settings like how long to capture timelapse photos. The app sends these parameters set by the user to the server which is running on the Portable system

described in Step 3. The Portable system is connected via USB to the camera taking pictures. After the exposures are taken, in Step 4 the user takes the SD-card storage from the camera and calibrates the images using the calibration matrix found in Step 1. Subsequently, software is run which will make an timelapse using the calibrated HDR images.

4.2 Color Calibration

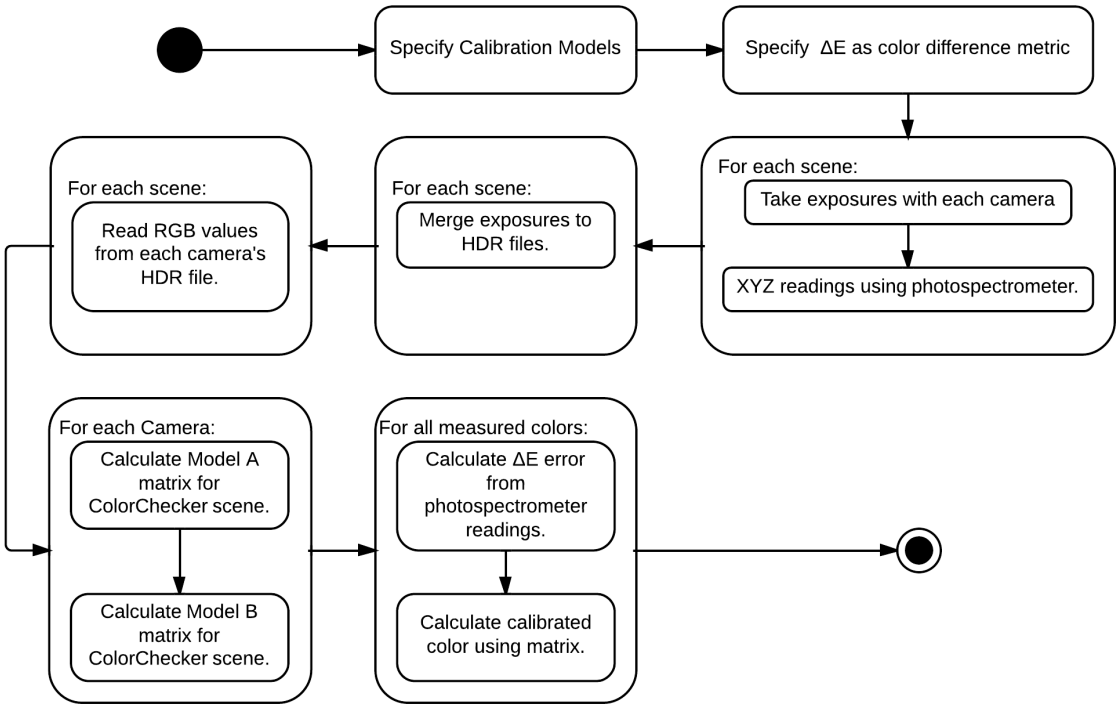


Figure 4.2: Activity Diagram of the Colorimetric Calibration

Figure 4.2 shows a brief overview of the calibration process, which is explained in more detail in the sections below.

4.2.1 Camera Models

As described in the Measurements section, the spectrophotometer values are extracted in XYZ colourspace and RGB values can be easily extracted from an HDR image using an a-priori written MATLAB function. To convert values in the RGB space to XYZ space, we utilised two models. The first model was:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{13} \\ m_{31} & m_{32} & m_{13} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.1}$$

The second model we used is an extension of the first one and also has an additional additive factor called the black level to capture signal noise.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \\ 1 \end{bmatrix} \tag{4.2}$$

The transformation matrix M is the 3x3 matrix in the first equation and the 3x4 matrix in the second equation. Since we have RGB values from the HDR image and XYZ values from the spectrophotometer, our task will be to find M. The above equations are in the form:

$$A \cdot x = b \tag{4.3}$$

where x is the equivalent of the transformation matrix M in our example. We can solve for x by using Equation 4.4.

$$x = A^{-1} * b \tag{4.4}$$

Alternatively, we can use solve this as a simple least-squares problem by doing this in MATLAB:

```
x = A \ b
```

The downside to this is that comparing the least squares error is not a good measure of perceived differences in colour or luminance. **Weber-Fechner Law** [52] states that the amount of change needed for sensory detection to occur increases with the initial intensity of stimulus, and is proportional to it [53]. The amount of change in luminance detected by the visual senses is dependant on and proportional to the initial luminance. In other words, absolute differences in least-square error is not a suitable error metric.
CIE 2000 ΔE colour difference formula [54] defines a metric so that the colour difference calculated between two values measured using colorimeters becomes close to the perceived colour difference of the human eye. We have therefore chosen CIE 2000 ΔE as the colour-difference metric which we will try to minimise, in order to find the best transformation matrix. The metric we use is given below:

$$E = \sum_{k=1}^{N} \Delta E(XYZ_m[k], XYZ_c[k]) + \lambda \left(log(Y_m[k]) - log(Y_c[k])\right)^2, \tag{4.5}$$

where $XYZ_m[k]$ are the XYZ colorimetric values of the colour patch k, as measured by the spectrophotometer. $XYZ_m[k]$ are the XYZ coordinates predicted from RGB values using the current

model M (right-hand-side of Equation 4.1 or 4.2). ΔE is the CIE2000 colour difference metric. The metric takes into account both change in colour difference i.e. ΔE and the squared difference in luminance. The squared difference is weighted by the $\lambda = 0.01$ term. This has been added because CIE 2000 ΔE assumes that the colorimetric values are relative, while we need to work in absolute luminance levels. E is used as an error function to find the model.

However, it is to be noted that the human visual system is more perceptive to some colour differences than others. A ΔE of 1.0 is often touted as the minimum difference between two colours which the human eye can detect but this is not accurate and varies from colour to colour as the human eye is more sensitive to certain colours than others [55]. In our experiment, we will not be solely depending on ΔE to validate our calibration's accuracy as some scenes are radically different to others (cf. "Library" and "Conference Room").

4.2.2 Measurements

Figure 4.3: Scenes 1: "ColorChecker" and 2: "Color patches". Numbers inside patches were not present in scene and were added later for identification and ease of use.

Scenes

We used four different scenes in the study. The first scene(Figure 4.3) contained a XRite ColorChecker Chart which was placed vertically by sticking to a wall. Adjacent to this, custom made chart containing "Color patches" was placed. There was no ambient lighting in the scene and the only sources of illumination were two photographic lights(Tricolor 1400W Ultra Cool Day Light), positioned at a 45 degree angle to the chart on both sides. The "ColorChecker" was used for calibration. The "Color patches" were treated as a separate scene to the "ColorChecker" and were used for testing only. Some of the patches in the custom "Color patches" were non-diffuse reflective i.e. they do not reflect incident light evenly and the observed colour is dependant on angle of observation.

Such patches were not used for measurements.

Besides these two scenes, we also used "Conference Room"(Figure 4.4) and "Library"(Figure 4.5)

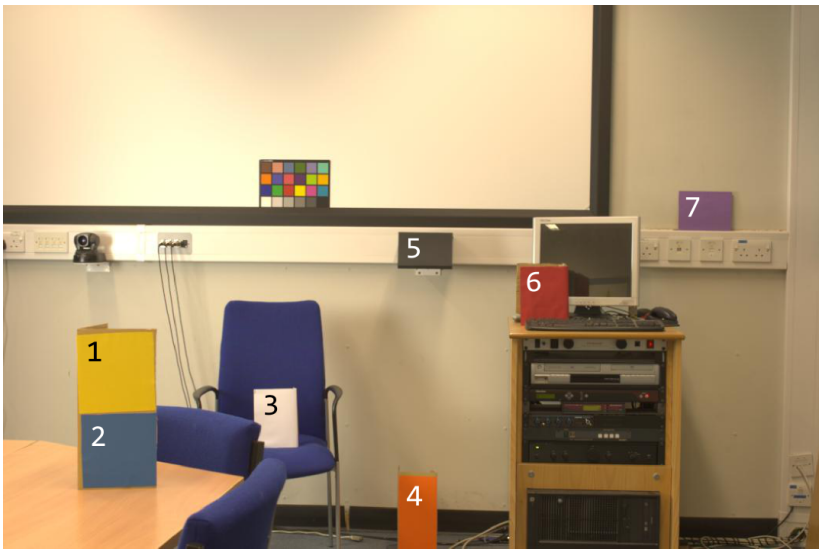


Figure 4.4: Scene 3: "Conference Room". Numbers were not present in scene and were added later.

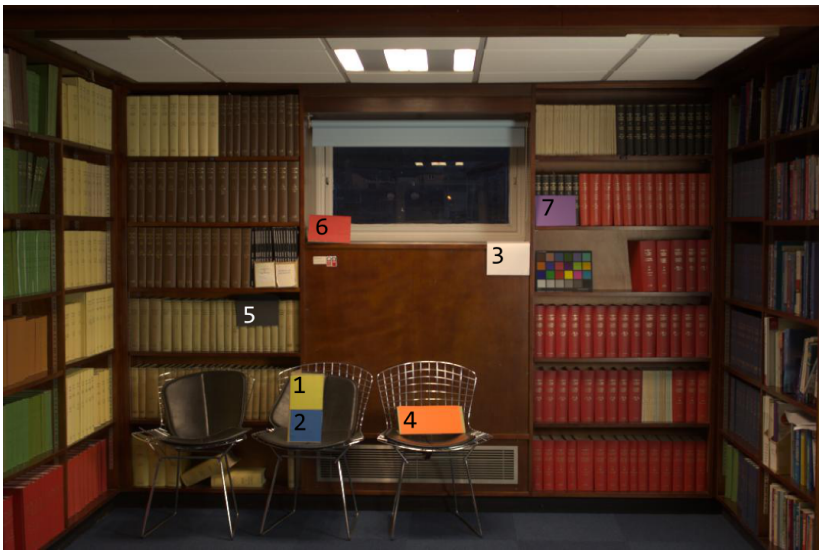


Figure 4.5: Scene 4: "Library". Numbers were not present in scene and were added later.

scenes. These rooms had their own distinct features. "Conference Room" was a well-lit room with plenty of ambient light whereas, "Library" scene was a dark scene. We switched on the room's lighting as well as using the photographic lights from the earlier scenes. For testing, we placed seven diffuse-coloured paper targets (made by sticking the paper to cardboard stands) in different positions. Once the scene was setup, no objects were changed or altered, so they remained constant across each

camera's exposures.

Cameras. Three cameras were used to photograph the scenes:

1. Canon 550D: An 18MP DSLR camera.
2. Canon 1000D: A 10.1MP entry level DSLR
3. Panasonic Lumix DMC-LX7: A 10.1MP point and shoot camera.

Scene	*Camera*	*Shutter speeds*	*ISO*	*Aperture*
ColorChecker	550d	1/16; 1/8; 1/4; 1/32; 1/64	100	f/5.7
	1000d	1/16; 1/8; 1/4; 1/32; 1/64	100	f/5.7
	Lumix	1/13; 1/50; 1/3.2	80	f/5.6
Conference	550d	1/5.2; 1/2.6; 1/1.3; 1/10.4; 1/20.7	100	f/8.0
	1000d	1/5.2; 1/2.6; 1/1.3; 1/10.4; 1/20.7	100	f/8.0
	Lumix	1/13; 1/50; 1/3.2	80	f/5.0
Library	550d	4; 8; 16; 32; 2; 1; 1/2; 1/49.4; 1/197.4; 1/1024	100	f/8.0
	1000d	4; 8; 16; 32; 2; 1; 1/2; 1/49.4; 1/197.4; 1/1024	100	f/8.0
	Lumix	1/13; 1/100; 1/1.6	80	f/5.0

Table 4.1: Capture settings for the exposures.

The experiments were conducted on a scene by scene basis. In each scene, the cameras were mounted on a tripod which remained in the same position. The target was to take captures at 5 different exposures at 1 f-stop interval in case of the Canon cameras and 3 different exposures at 2 f-stop intervals for the Panasonic camera. Each stop increment essentially allows twice as much light to hit the CMOS sensor (or half as much if going down). This is how we capture areas which would be usually underexposed or overexposed in a standard Low Dynamic Range photograph. Due to the very high contrast in the "Library" scene, we decided to capture 10 exposures to avoid any overexposed or underexposed pixels. We let the camera's metering choose the optimal exposure value and then using this as the base exposure, the other exposure values were the next 2 increments/decrements for the Canon cameras, and for the Panasonic, it was the increment/decrement after a 2 f-stop jump. Exposure values for each scene and each camera used for that scene is shown in Table 4.1. Photographs for the Canon cameras were captured using remote capture from a computer. A USB cable was connected from the cameras to a computer running Linux. The camera would be mounted on the tripod, turned on and set in Manual mode. A shell script which took exposure values as the input arguments was run with the appropriate exposure values. The shell script would then call gPhoto2 and capture images with the arguments provided as the exposure values. A sample call for this shell script is:

```
gphoto_executor 1/16 1/8 1/4 1/32 1/64
```

Figure 4.6: Cameras used for calibration training and testing.

The code is provided in Appendix. The Panasonic camera is not supported by gPhoto2 however, it did have built-in auto exposure bracketing, so using that we set the required EV stops on the camera and took each exposure set.

XYZ Measurements.

The XYZ values against which Colorimetric Calibration is based are measured using a Specbos 1211 spectroradiometer. The measurements are taken on a scene by scene basis after the exposures are captured using the three cameras. The radius of the spectroradiometer can be adjusted so it is within the bounding box of the colour patches of the ColorChecker. Using a simple MATLAB function, we could take the required number of luminance readings and save them in a text file. An average of 5 readings was taken for each measurement. This was repeated for each scene for a total of four sets of readings. The spectroradiometer measures values in the CIE Yxy colour space (which is a derivative of the XYZ colour space). The values can be easily converted to CIE XYZ colour space given the Yxy values. A chromaticity diagram which shows the measured colours in all four test scenes is shown in Figure 5.4

HDR Image Creation.

As a result of using 4 scenes, and 3 cameras to shoot with, there are 12 sets of exposures. Each

set of exposures is merged using *pfstools* [56] software. *pfstools* is a set of command line software for reading, writing and manipulating HDR images. Each exposure set is kept in a separate directory. For each of the 12 sets, the folder is opened in shell and the exposures (RAW images) are passed as input arguments to pfsinme which can merge the RAW images into a single HDR file. For example, after opening the folder containing the exposures to be merged in shell, we can run the following command to merge and create a single HDR image:

```
pfsinme *.CR2 | pfshdrcalibrate -r linear -v --bpp 16 | pfsout 550d_cc1.hdr;
```

The pfsinme command reads in the exposures before piping it to pfshdrcalibrate which has the arguments that specify it to use a linear response (-r) and to use 16 bits per pixel (-bpp 16). This is then piped to pfsout which writes to an hdr format file. Internally, pfstools calls a software called `DCRAW` [57] with the parameters `-c -o 0 -4 -w`. -c tells DCRAW to write the image data to standard output (which can be piped through to the other pfstools processes e.g. `pfshdrcalibrate`). `-o 0` specifies raw output colour space. `-4` tells DCRAW to write 16-bit linear image instead of the 8-bit gamma corrected which it would usually. `-w` specifies that the camera white balance should be used.

pfstools also has an interface which can be accessed from *MATLAB*. This enables us to read the HDR file in MATLAB and read the RGB values of each colour patch in the image. The user clicks on the centre of the patch and the script then computes the average colour within that patch.

4.3 Remote Camera Capture

An overview of the remote capture and timelapse generation is represented in Figure 4.7.

To initiate remote capture of the exposures, an Android Application interface was used. Having remote control as a feature adds ease-of-use and portability to the system. Once the camera is turned on and connected to the server (which is publicly accessible by its IP-address), the android application can communicate wirelessly with the server.

The application was created using Eclipse IDE and deployed on a Samsung Galaxy S3 (GT-19305), running Android OS version 4.3, for testing purposes. The target Android OS version was 4.3 (Jelly Bean), however this app is backwards compatible up to 3.0 (Android Version Ice Cream Sandwich).

Android apps are written in the Java programming language, although there are some notable differences in its implementation in the Android API. Primarily, Android utilises a android-bespoke virtual machine called Dalvik instead of the Java Virtual machine(JVM) used by Java [58] [59]. Dalvik also cannot handle tasks like custom class loading and runtime bytecode generation. There is no Just In Time (JIT) compilation either.

The app is called *RemoteTimelapse* and when started, it shows a screen containing a text-entry box (Figure 4.8a) asking for the IP-address of the server running on the Raspberry-Pi. There is a checkbox to remind the user to set up the remainder of the apparatus properly before entering the address (a server connected to a camera ready to take exposures). The user taps the *Submit* button

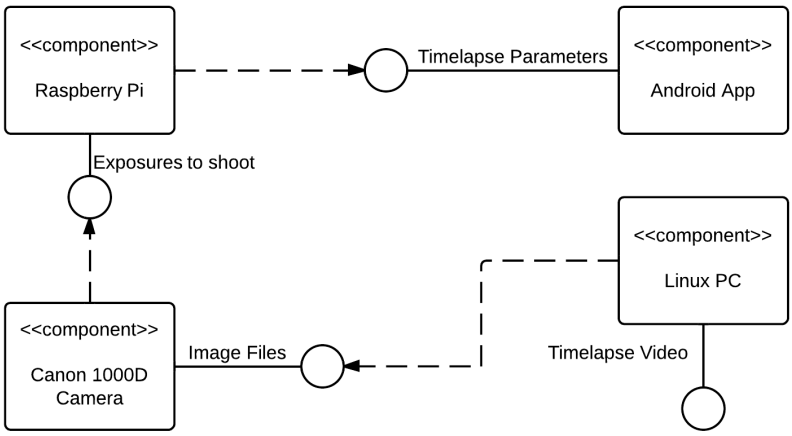
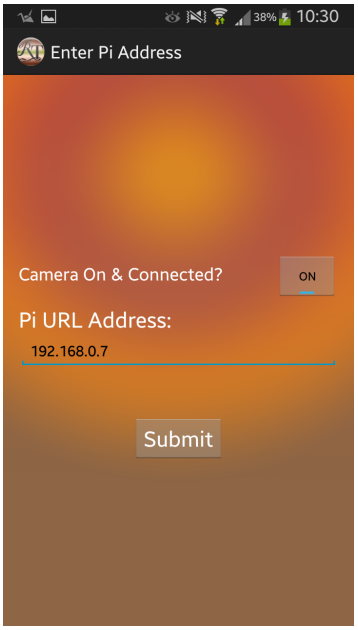
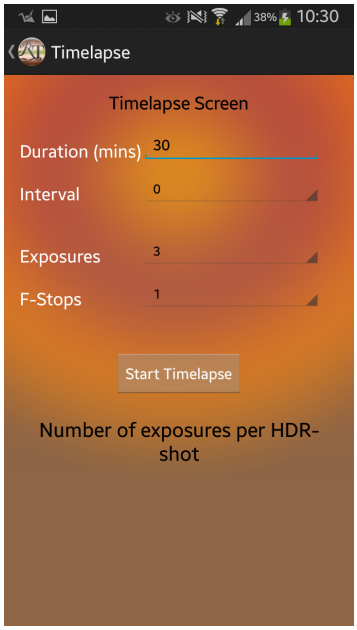


Figure 4.7: Component Diagram of the Timelapse generation system

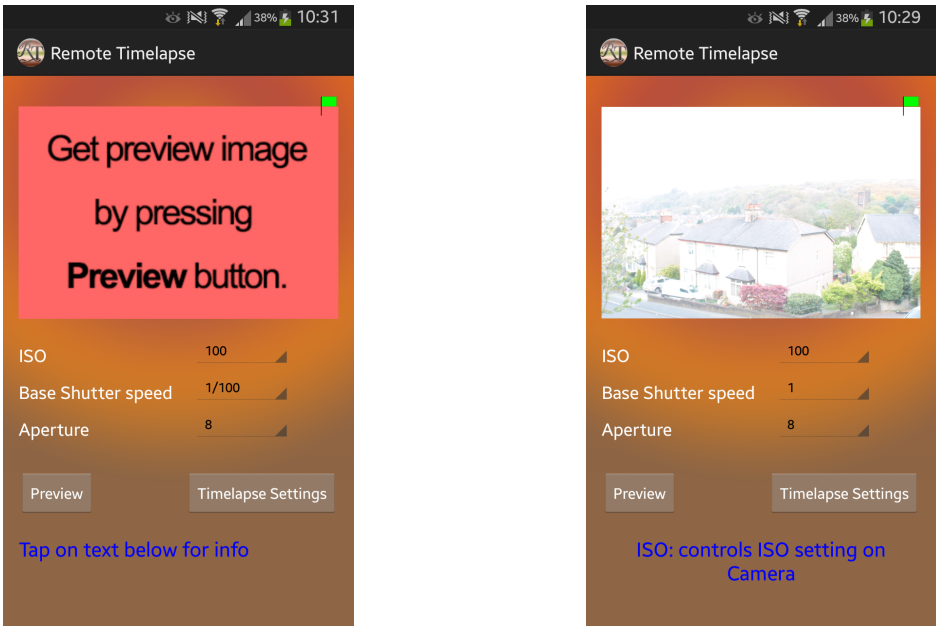


(a) Startup Screen



(b) Start Timelapse here.

Figure 4.8: Address Entry screen and Timelapse screen



(a) Preview Screen Initially

(b) After pressing Preview button

Figure 4.9: Test the camera settings and take previews.

once they have finished both fields, which takes the user to the *Preview* screen.

When the user enters the *Preview* screen(Figure 4.9a), they can choose the capture parameters and get a live preview (Figure 4.9b) to check how the camera captures the scene using the selected settings. The screen also contains a help-section at the bottom. For example, if the user wants an explanation of the ISO field, tapping on the ISO label shows a descriptive hint at the bottom. Each time the Preview button is pressed, the app sends the capture parameters to the server where a program uses these to take a preview image and store it on the server.

The app then retrieves this image and shows it to the user. The user can set the ISO, Shutter-speed and Aperture values. After taking a preview image, if the user is unhappy with the camera's output (for example, if the image is too dark) then the user can adjust the shutterspeed or ISO to a higher value. Any number of previews can be taken to refine the settings. Once a desirable preview image is shown, the user can then proceed with the timelapse by pressing the *Timelapse Settings* button, which takes the user to the next screen.

On the Timelapse Screen (Figure 4.8b), variables with regards to HDR Timelapse video capture can be set. The Duration setting controls for how long the camera will keep taking shots i.e. if the user enters 30, the camera will take HDR exposures repeatedly until 30 minutes have elapsed. Exposures controls how many camera shots will be used to merge into each HDR image. For example, if the user chooses 3 exposures, the camera will take 3 photos for each HDR frame in the final timelapse video. The Interval field can be used to set a time interval between each HDR-shot. This can be used in combination with the Exposures setting to take timelapses of slow phenomenon. For example, if exposures is set to 3 and interval is set to 60, the program will take 3 exposures (for each HDR

image) and then wait 60 seconds before it will take the next set of exposures. This needs to be used in conjunction with a suitably long Duration setting for best results.

F-Stops determines the number of jumps to take from one shutterspeed value to the next for each exposure in the HDR image. The default value is 1, so for a set of 3 exposures, using the base shutterspeed value from the Preview screen, the other two exposures are the next available shutterspeed on the camera either side of the base shutterspeed. When a F-Stops value of 2 is chosen, instead of the next available shutterspeed (jump of 1), the shutterspeed value after skipping the next available one is chosen (jump of 2). After setting these four variables, the user starts the exposure capture process by pressing the *Start Timelapse* button.
The phone-server communication is achieved using POST, a Request Method defined by the HTTP protocol [60]. The HTTP POST method allows the App to send blocks of data to a data-handling process on the server. For the **Preview** screen, when the button is pressed to retrieve a preview from the camera, the App sends the ISO, Shutterspeed and Aperture values via POST to the PHP-scripted *preview.php*. On the **Timelapse Sceen**, when the user presses the button to start a timelapse, the ISO, Shutterspeed, Aperture, Exposures, Duration, F-Stops and Interval parameters are send via HTTP-POST to *timelapse.php* residing on the server. The PHP files parse the incoming parameters and do the appropriate functions. This process is further explained in Section 4.4.

4.4 Portable Server System

Figure 4.10: Raspberry Pi [3]

The Raspberry Pi is a small credit card sized (8.5cmx5.6cm) single-board computer made by an educational charity called Raspberry Pi Foundation [61]. For this dissertation, Raspberry Pi Model B was used, which has 512MB RAM, 2 USB ports and a 700 MHz ARM11 processor. With its small weight of 45g, it is highly portable. Due to its meagre power requirements of 700 mA (3.5 W), it can

also be powered from a battery-pack.

The Pi was configured to run Raspbian OS, a Debian based Operating System specifically optimised for the limited resources of the Pi's hardware. All these features made the Pi an ideal component for the role of the central server in the HDR capture system. Due to its limitations in processing power (it has a paltry 700MHz processor and a 512MB RAM) and requirements of the software used to generate HDR images (*pfstools* [56]), exposure-merging (merging multiple exposures to generate a single HDR image file) has to be done on an external PC.

For the purpose of developing the system, a LAMP solution stack derivative was used. For a typical web application, LAMP is one of the most popular choices. Each letter in the acronym refers to the layers of the system. [62]
L = Linux (Operating System)
A = Apache (Web Server)
M = MySQL (Database Management System)
P = PHP/Perl (scripting languages)

For our purposes, Lighttpd was used instead of Apache web server. All the software used is open-source and freely available.

Lighttpd was chosen as an alternative to Apache because as the name implies Lighttpd is a lightweight webserver capable on working smoothly on the limited processing power of the Raspberry Pi [63]. MySQL was not installed as it is not needed. PHP version 5 was also installed.

In the appendix, *preview.php* and *timelapse.php* are provided. These are the PHP scripts accessed by the Android App; *preview.php* for capturing a camera preview and *timelapse.php* for starting the timelapse capture. *preview.php* parses the incoming parameters and uses them to capture a preview photo using the camera attached to the Pi. It does this by issuing a call to a shell script (*gphoto2_reset.sh*) which in turn calls *gPhoto2*, a command-line tool used to remote control cameras from a linux shell. This is explained in more detail in Section 4.4.2.

4.4.1 Timelapse-capture scripts

To make the process of HDR capture, color calibration of images and timelapse generation a smooth process, shell scripts were used. They automate the process, eliminating the need to write repetitive commands every time a colour calibrated HDR timelapse needs to be generated.

gPhoto2_reset.sh

gphoto2_reset.sh is a simple workaround to a bug in *gPhoto2* which causes the camera to stop responding randomly after a remote-controlled capture. As part of the workaround, the camera USB port is reset before and after each call, hence eliminating the bug.

An example gPhoto2 program call to capture an image to the camera's storage after setting the shutterspeed to 1/00 and ISO to 100 is:

```
gphoto2 --set-config=shutterspeed 1/100 --set-config=ISO 100 --capture-image
```

gphoto2_reset.sh, the bug workaround, works similarly:

```
gphoto2_reset.sh --set-config=shutterspeed 1/100 --set-config=ISO 100 --capture-image
```

timelapse.sh

timelapse.sh takes the exposures required for the HDR files.

hdrgenerator.sh

This shell script merges the exposures taken by *timelapse.sh* into an HDR file and colorimetrically calibrates them. The HDR files are then tonemapped and saved as Low Dynamic Range JPEG files. The JPEG files are used to generate the timelapse video.

4.4.2 *gPhoto2* Remote Capture

Capturing the timelapse requires a call to *timelapse.sh*. This is called by the server when it receives the parameters for the timelapse capture from the android app. *timelapse.sh* takes as input arguments the parameters, and then calls the *gphoto2_reset.sh* to take the appropriate exposures. The timelapse shell script is called using:

```
$out=system("sudo -u pi /home/pi/scripts/timelapse.sh -i $iso -s $shutterspeed
-a $aperture -d $duration -e $exposures -l $interval -f");
```

system is a PHP function to run an external program, and the output from this function is assigned to variable *out*, which can be used for debugging purposes later. *sudo -u pi* ensures the command following it is run as the user pi (who has more privileges than the alias of the web server). The *timelapse.sh* is run with the following options: -i (ISO), -s (SHUTTERSPEED), -a (APERTURE), -d (DURATION), -e (EXPOSURES), -l (INTERVAL). The variables next to the switches contain the respective values which have been obtained from the HTTP-POST input. The logic in *timelapse.sh* can be summarised in a flowchart as shown in Figure 4.11.

timelapse.sh checks the mode that the camera dial is set to. For HDR-capture to work, the camera needs to be in Manual or Aperture-Priority mode (abbreviated as M and Av on the camera. Refer to Section 2 for more information.).

If the camera is in Av mode then *timelapse.sh* only sets the initial ISO and Aperture settings on the camera, the camera will automatically set the shutterspeed values. On the other hand, if camera is in Manual mode, *timelapse.sh* will determine the shutterspeeds to shoot based on the base shutterspeed sent by the Android App.

4.5 HDR Merging

Once the duration specified by the user has elapsed, the server will stop taking captures for the timelapse. The next step involves creating calibrated HDR images from each set of exposures. Finally, the HDR exposures are merged to create a video. HDR merging is done using a set of tools called

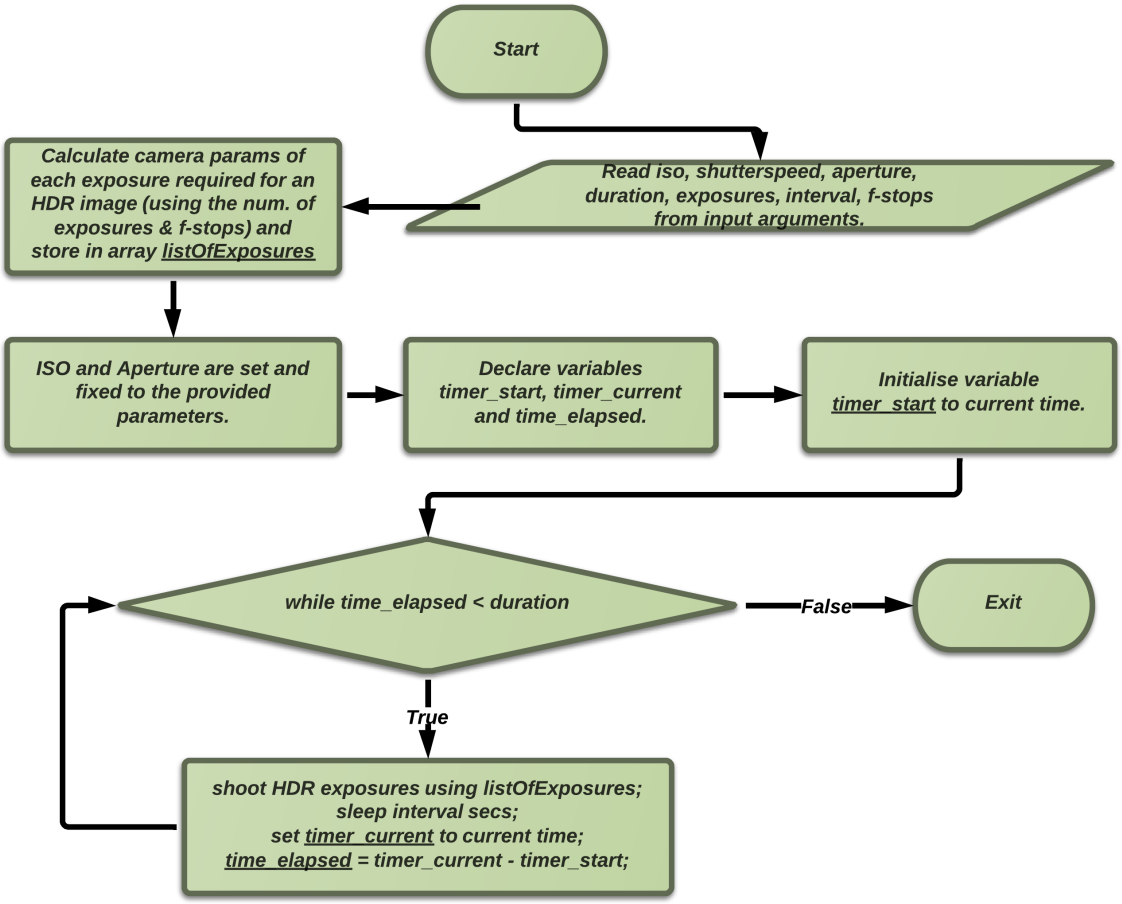


Figure 4.11: *timelapse.sh* design.

pfstools [56]. Unfortunately, the limited 512MB ram of the Raspberry Pi cannot support HDR file creation since this involves loading several large camera RAW files into memory. Therefore, for this step another computer is used. pfstools has been tested under various unix distributions and thus any unix based computer which meets the minimum hardware requirements of pfstools should suffice. For this step, a HP Pavilion g6 laptop was used. It has 6GB RAM and a 2.40GHz processor and is running elementary OS (an ubuntu-based distribution).

4.5.1 Calibrated HDR Images

The camera is connected via USB to the HP laptop when the user is ready to generate the timelapse video from the captured exposures, once the server has finished taking the captures. To make the process easier, a shell script called *hdrgenerator.sh* was created. This script can do a number of tasks depending on which of the following options are passed through as an input argument when executing

Figure 4.12: Canon 550D connected to server running on Raspberry Pi.

the script:

1. **-e (exposures)**: Specify the number of exposures used per HDR shot. The default is 3.
2. **-d** : Download all images from the camera
3. **-m** : Create the calibrated HDR files.
4. **-c** : Create the timelapse video.

When *hdrgenerator.sh -d* is called, the script executes

```
gphoto2 -D
```

which is a quick gphoto2 command to download all images on the sd-card.

Each set of exposures (by default 3 exposures are used, unless specified otherwise using the -e option), are then processed iteratively using *pfstools*. Using the *-m* option of *hdrgenerator.sh*, *pfstools* is automatically called on the exposures:

```
pfsinme "file1.CR2" "file1.CR2" "file1.CR2" | pfshdrcalibrate -r linear -v | pfscolortransform
 --xyzrgb matrixFile.txt | pfssize -x 1280 -y 800 | pfstmo_mantiuk06 | pfsout HdrImage.jpg
```

where *pfsinme "file1.CR2" "file2.CR2" "file3.CR2"* reads in multiple exposures which are to be merged into a single HDR image. *pfshdrcalibrate -r linear* creates an HDR image from the data piped

to it from *pfsinme* using a linear response curve. This ensures no correction is applied to the exposures and they are exactly as captured on the camera sensor. *pfscolortransform –xyzrgb matrixFile.txt* reads in a 3x3 matrix from the file "matrixFile.txt" which will be used to calibrate the HDR image. The matrix contained in the file is the calibrated matrix derived from the colorimetric calibration function trained in Section 4.2.

pfssize -x 1280 -y 800 simply resizes the frames to a resolution of 1280 x 800. *pfstmo_mantiuk06* calls the mantiuk06 tone-map operator [64]. A tone-map operato is used to portray the HDR image created using *pfshdrcalibrate* within the displayable range of low dynamic-ranged devices. The mantiuk06 operator evens out the areas in the image with high contrast differences. *pfsout HdrImage.jpg* writes the tone-mapped colour-calibrated HDR data to a JPEG file.

4.6 Timelapse Video

At this stage we have the JPG files in a directory, which are colour-calibrated HDR files. Running the hdrgenerator.sh script with the -c option generates a timelapse video in the current directory. From the shell the call is:

```
hdrgenerator.sh -c
```

The script file is essentially running the following command when it is invoked:

```
mencoder  mf://*.jpg  -mf fps=24:type=jpg -oac copy -ovc xvid -xvidencopts
 fixed_quant=1:chroma_opt:vhq=4:max_bframes=1:quant_type=mpeg:threads=8
 -o output.mp4
```

mencoder is a freely available command line tool to encode images to a timelapse video. The *-mf fps=24:type=jpg* option tells mencoder the number of frames to use per second of video and the type of image files used for the frames. A frame rate of 24 is used as that is the industry standard. XviD is chosen as the video compressor (*-ovc xvid*).

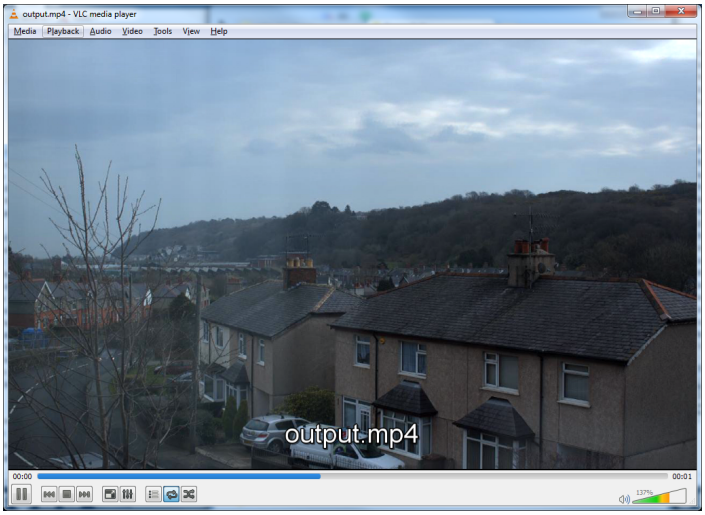


Figure 4.13: Timelapse video generated using *mencoder*

Chapter 5

Evaluation & Results

5.1 Results Of Color Calibration

To test the validity of the calibration, we will use the camera models described in Section 4.2.1 and check the error rate of the transformation matrix M. Using the ColorChecker scene (Scene 1), we computed two matrices for each camera: one each for the models in Equations 4.1 and 4.2. This gives us a total of 6 matrices for the ColorChecker since we have three cameras. Substituting the matrix and the recorded RGB colours into equations 4.1 and 4.2, we can derive each camera's XYZ calculations. The next step is to compare these set of XYZ's with the photospectrometer's XYZ values using the CIE2000 ΔE colour difference formula. This step has been presented in tabular form in Table 5.1. The last three columns represent the ΔE errors for the cameras compared to the reference XYZ from the photospectrometer.

We calculated the ΔE for both models in all scenes and the differences were small except in the "Library" scene, so for the sake of brevity, we have only included Model A(Equation 4.1) in the Tables. For the testing scenes "Conference Room" and "Library", Model B is of a poorer quality than Model A. We are unsure as to the cause of this as we expected the extra black level in the model to capture noise error. The poor predictions are probably caused by overfitting. The cause may be that the number of samples provided by the ColorChecker (24) simply isn't enough data for the model to make accurate predictions with, and it needs a bigger data set to classify accurately. Thus, if we are to base calibration simply on Scene 1 i.e. the ColorChecker, then the simple model without a black level will be enough.

Averaged results for Model B(Equation 4.2) are presented at the bottom of the tables for comparison. In Table 5.1, four colours are also displayed. Column M displays the reference colour, picked up by the photospectrometer in its native colour space and converted to the sRGB colour space using a Rec. 709 transformation matrix. The ΔE_{num} columns report the error of the uncalibrated cameras and the C_{num} columns display the colours after colorimetric calibration of the camera images. Note that the colours shown have been converted into sRGB colour space without correcting for white balance. However, perceptual differences (if any) should be visible despite not allowing for white

#	M	C_1	C_2	C_3	$\Delta E_1 1000D$	$\Delta E_2 550D$	$\Delta E_3 Lumix$
1					0.2 (0.0)	0.1 (0.1)	0.8 (1.7)
2					1.1 (2.5)	1.2 (3.6)	1.1 (2.6)
3					0.9 (1.8)	0.7 (2.0)	0.1 (0.3)
4					0.6 (1.2)	0.4 (0.7)	1.1 (0.4)
5					1.0 (0.1)	0.8 (0.2)	0.3 (0.4)
6					0.6 (1.2)	0.7 (2.0)	1.4 (4.5)
7					1.1 (2.3)	0.3 (1.2)	0.6 (1.8)
8					0.2 (0.9)	0.6 (0.6)	0.4 (2.1)
9					0.3 (0.3)	0.4 (0.5)	0.4 (0.4)
10					0.5 (0.1)	0.6 (0.7)	0.5 (1.2)
11					0.3 (0.6)	0.4 (1.0)	0.0 (0.0)
12					0.9 (3.5)	1.1 (3.7)	0.7 (0.7)
13					0.5 (4.1)	0.9 (3.3)	0.5 (4.2)
14					0.5 (2.3)	0.3 (1.1)	1.3 (0.3)
15					0.7 (2.4)	0.5 (1.6)	1.2 (3.9)
16					0.4 (0.6)	0.7 (2.1)	0.7 (0.8)
17					0.6 (1.9)	0.5 (1.4)	0.7 (0.5)
18					1.1 (5.0)	1.6 (5.4)	2.1 (7.7)
19					0.6 (2.2)	1.1 (1.2)	0.4 (1.8)
20					0.2 (0.2)	0.4 (0.0)	0.3 (0.1)
21					0.6 (0.5)	0.1 (0.0)	0.5 (1.6)
22					0.4 (0.4)	0.7 (0.1)	0.7 (0.2)
23					0.7 (0.5)	0.6 (0.0)	0.7 (3.1)
24					0.6 (3.8)	0.5 (0.0)	1.8 (13.5)
Averaged Model A					0.6 (1.6)	0.6 (1.4)	0.8 (2.2)
Averaged Model B					0.7 (2.1)	0.8 (1.9)	0.7 (1.8)

Table 5.1: Calibration results for the ColorChecker. M column shows the colour as measured by the spectrometer (after converting to the sRGB colour space). C columns show the colour after camera colorimetric calibration. ΔE is CIE2000 DeltaE colour difference between the colour measured with the photospectrometer and the colour captured with a camera . The numbers in parenthesis are the relative error in luminance in percent ($\Delta L/L$). The errors for each colour patch are provided only for Model A (Equation 4.1).

#	M	C_1	C_2	C_3	ΔE_1	ΔE_2	ΔE_3
1					0.5 (1.1)	0.7 (0.8)	0.8 (2.3)
2					0.4 (0.8)	0.7 (1.4)	1.0 (2.1)
3					0.8 (1.2)	0.5 (0.1)	1.8 (7.5)
4					0.8 (1.8)	0.4 (1.4)	1.0 (4.7)
5					1.3 (5.4)	1.1 (4.6)	1.8 (2.3)
6					0.5 (1.2)	0.5 (1.6)	0.5 (1.4)
7					0.5 (1.9)	1.0 (3.6)	1.2 (0.9)
8					0.5 (0.3)	0.9 (2.7)	1.7 (0.7)
9					1.0 (4.3)	1.2 (4.9)	0.4 (1.0)
10					1.3 (3.6)	1.7 (3.0)	2.6 (1.3)
11					1.1 (3.9)	1.2 (4.7)	0.5 (1.2)
12					1.2 (3.6)	1.7 (6.3)	1.1 (2.8)
13					0.6 (1.7)	1.3 (4.7)	1.8 (1.2)
Averaged Model A					0.8 (2.4)	1.0 (3.1)	1.2 (2.3)
Averaged Model B					0.8 (2.4)	1.0 (2.8)	1.2 (2.2)

Table 5.2: Calibration results for the "Color patches" scene. The labels are the same as in Figure 4.3.

#	M	C_1	C_2	C_3	ΔE_1	ΔE_2	ΔE_3
1					0.8 (2.4)	3.2 (12.4)	1.5 (4.5)
2					1.4 (5.1)	2.7 (12.5)	1.9 (9.8)
3					1.1 (4.5)	3.0 (11.7)	2.4 (8.7)
4					1.6 (7.1)	2.2 (8.6)	2.6 (9.8)
5					2.1 (12.2)	2.5 (9.3)	1.9 (12.4)
6					2.8 (15.6)	2.2 (8.5)	2.8 (15.5)
7					0.5 (0.3)	4.6 (19.8)	0.9 (2.5)
Averaged Model A					1.5 (6.8)	2.9 (11.8)	2.0 (9.0)
Averaged Model B					1.2 (3.8)	3.2 (14.3)	2.5 (4.7)

Table 5.3: Calibration results for the "Conference room" scene. The labels are the same as in Figure 4.4.

#	M	C_1	C_2	C_3	ΔE_1	ΔE_2	ΔE_3
1					4.1 (10.9)	5.3 (7.4)	1.3 (4.8)
2					1.1 (5.9)	1.5 (8.0)	0.6 (0.4)
3					2.8 (7.6)	3.6 (5.2)	2.7 (4.5)
4					4.2 (14.8)	1.8 (5.6)	3.0 (8.3)
5					0.7 (2.9)	0.7 (8.6)	0.7 (3.4)
6					0.8 (1.9)	2.0 (0.6)	1.4 (0.6)
7					2.2 (8.7)	0.7 (0.8)	1.4 (5.5)
Averaged Model A					2.3 (7.5)	2.2 (5.2)	1.6 (3.9)
Averaged Model B					4.5 (17.3)	3.6 (5.5)	4.4 (14.0)

Table 5.4: Calibration results for the "Library" scene. The labels are the same as in Figure 4.5.

balance correction. The ΔE values in the tables higher than 4.0 is marked in red colour.

Observing the calibration tables after calibration, we can see that there are only small differences in the colours. In the first testing scene "Color patches"(Table 5.2), all three cameras had relatively small colour difference compared to the reference colour. The highest error was for the Lumix in patch 10 (Orange) with ΔE=2.6. The lumix also had the highest average error in that scene too. The average error was small (ΔE=1.2). Canon 1000D had the least colour difference from the photospectrometer readings with a ΔE of 0.8. Another point of note is the similar Average ΔE errors for both models for all three cameras. This is where the average luminance values (in parenthesis) helps to differentiate objectively the effectiveness of the models.

Looking at Tables 5.3 and 5.4 we see that ΔE is much higher. The average ΔE is 2-3 times higher than for the "Color patches" scene. The Canon 550D specifically did much worse than the other cameras in the "Conference Room" scene. We got a high ΔE of 4.6 for Patch 7. The explanation for this might lie in the very high luminance difference of 19.8 observed on this patch. This was unexpected because the shots taken with the preceding and succeeding cameras yielded very good results. The Canon 550D and Canon 1000D also shared the same capture parameters as can be seen from Table 4.1. What makes the error harder to isolate is that the 550D was significantly better than the other cameras for some patches (Patches 5 and 6). It is to be noted that although these numbers represent numerically higher errors, a ΔE of 4 is considered a Just Noticeable Difference(JND). So even in the case of the few patches were ΔE is high, they are only just perceptible. Comparing the photospectrometer and post-calibration colours in 5.3, the differences are unnoticeable and look very similar visually. The colour differences in the "Library" scene as seen in Table 5.4 were reasonably small. Model A showed small error especially for the Lumix, though as noted earlier, Model B gave a high error metric for all three cameras.

The average luminance values in "Conference Room" and "Library" were inconsistent for the three cameras. Canon 550D fared worst in "Conference Room" whereas in the "Library" scene, it was the nearest to measured luminance. As mentioned before, this could be attributed to the complex nature of the scene. The relative luminance in these scenes were high (up to 15% for some colours). We also found that for scenes with well controlled illumination and no external lighting variables ("ColorChecker" and "Color patches"), the error in luminance was mostly below 5%. A notable exception to this was the luminance difference in the colour Black (Patch 24) in the "ColorChecker" scene as captured by the Lumix, which is a high value at 13.5%. In the complex scenes, White & Orange (Labels 3 & 4) were the most problematic colours. Visually comparing the colours measured in Table 5.3 with the colours in Table 5.4 shows perceivable difference, especially observed in the Black (#5) and Yellow (#1) colours. This highlights the strong role illumination and shading had in terms of colour measurement, due to the lighting conditions in both scenes being completely different, as explained earlier.

Figures 5.1, 5.2, 5.3 show the cameras' images pre/post calibration. The differences between the images in the first row is in stark contrast with the similarity of images in the second row. However comparing the images before calibration to any image post-calibration, we can say from the visually perceived differences that all three cameras needed calibration to produce colours close to

Figure 5.1: Comparison of images before and after calibration for the "Color patches"scene.

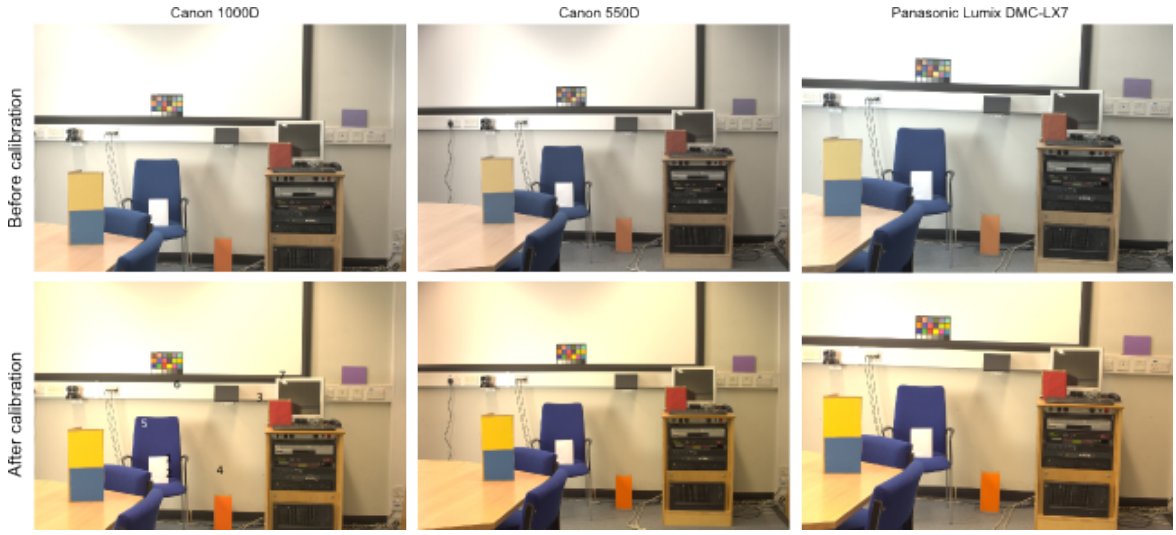


Figure 5.2: Comparison of images before and after calibration for the "Conference Room"scene.

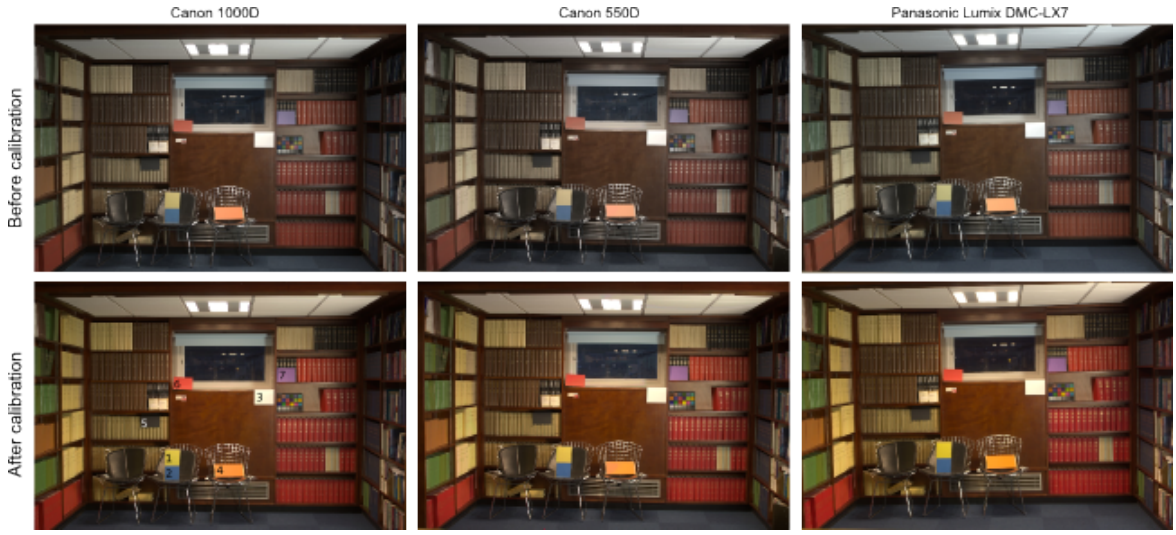


Figure 5.3: Comparison of images before and after calibration for the "Library"scene.

the captured colours of the photospectrometer. The results can be summarised visually in Figure 5.4. It displays the measured colours in terms of their chromatic coordinates in a 2D representation of the CIEXYX space. Our transformation matrices contained negative values which is displayed in the figure by the points which lie outside the sRGB and Adobe RGB colour gamuts. We could not find an explanation for the negative values, as they indicate that some primaries lie outside the visible gamut. The camera sensor can also capture infrared spectrum of light that is invisible to the human eye. Most digital cameras contain an infrared filter that blocks most of the infrared rays that would otherwise hit the sensor [65]. This could be a cause of the negative values in the calibration matrix.

5.2 Evaluating Implementation

5.2.1 Remote Capture using *gphoto2*

The program used to remote capture, *gphoto2* introduces time overhead compared to a manual shutter-press. Whereas manually pressing the shutter button takes a capture nearly instantaneously, *gphoto2* takes several seconds. For an HDR shot comprising of several exposures, this means there is a time delay between shots. *gphoto2* provides two options for direct capture of photos: *–capture-image* takes a photo and stores it on the camera's internal sd-card. *–capture-image-and-download* makes the camera take a photo and then downloads the photo to the computer, deleting the photo from the camera. Due to the additional I/O involved in transferring the photo after capture, *–capture-image-and-download* is expected to take longer than *–capture-image*. For comparison, *–capture-image* and *–capture-image-and-download* were tried after setting the image format on the camera to a small JPEG file. The apriori assumption is that since the RAW file is considerably larger than the JPEG file (circa 10MB vs circa 0.5MB), the *gphoto2* calls will be considerably faster for the smaller format. This is for testing purposes only because as discussed previously, JPEG format is not suitable for this

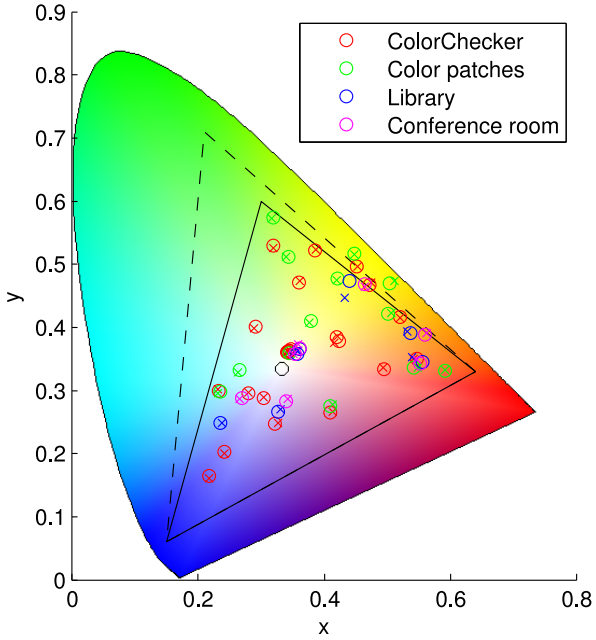


Figure 5.4: Chromatic coordinates of all measured colours for four test scenes. Circles correspond to photospectrometer measurements and crosses to the calibration results for Canon 550D. Calibration for other cameras produced similar results. The solid line denotes the sRGB colour gamut and dashed line the Adobe RGB gamut.

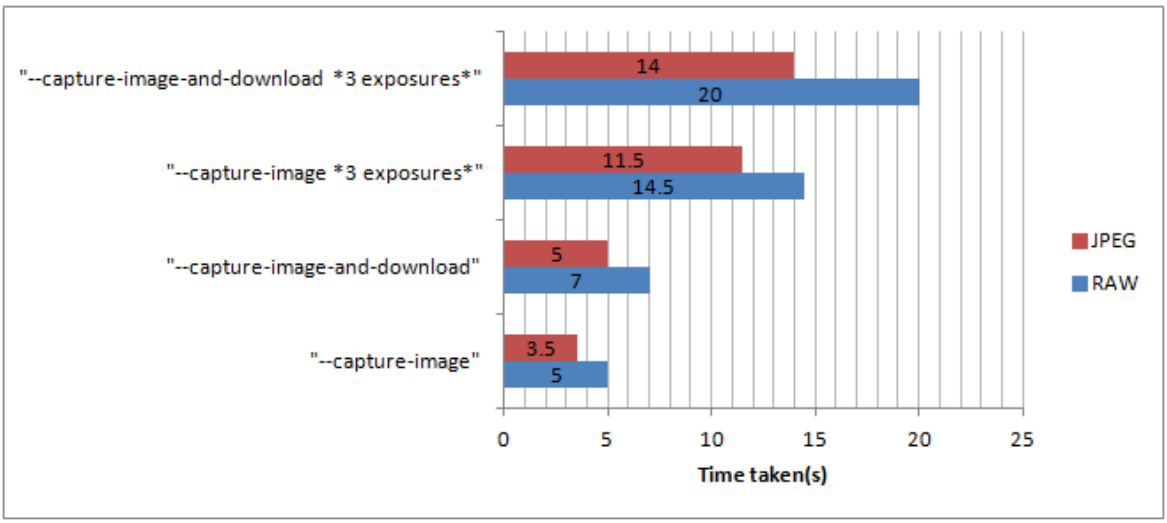


Figure 5.5: Timings for the *gphoto2* calls.

dissertation.

Figure 5.5 show the time it takes to execute each command given in each row, in both JPEG and RAW formats. The 3 exposures used for this measurement were shutterspeed values of 1/100, 1/80 and 1/200. When measuring the time it takes to take three exposures, the above chart is only representative of these three shutterspeeds. For other shutterspeeds, different values timings should be expected, as the shutterspeed determines how long the camera's sensor captures light. Theoretically, a shutterspeed of upto 60 seconds is possible for the Canon 1000D.
It can be observed that there is only a small difference in using a small JPEG file as opposed to a RAW file. Also a single HDR shot spans an time period of 14.5 seconds, which means scenes with noticeable change during this timespan will be problematic, introducing issues like **ghosting**.

An alternative method of capturing exposures was researched involving Aperture-Priority mode on the camera. So far, this project has been working on the assumption that the camera will be set in Manual mode prior to starting remote capture. This is because Manual mode allows full configurability of all the required parameters of the camera (ISO, shutterspeed, aperture). In Aperture-Priority mode, the user can only set the Aperture and the ISO. The camera automatically sets the shutterspeed based on its metering [66].

In principle, this is an ideal setup to use for HDR photography, as once the first exposure has been taken in Aperture-Priority mode, using a setting called *exposurecompensation* which can be adjusted via *gphoto2*, multiple exposures can be taken. This is a much better solution than the default implementation as the camera selects the best possible shutterspeed based on the illumination in the scene and the pre-selected aperture. Timelapse videos generated using captures taken via Aperture-Priority mode were superior to timelapse videos generated using pre-selected shutterspeeds in Manual mode.

Finally, Auto Exposure Bracketing (AEB), a widely used technique for HDR photography, was also tested. In AEB mode, on each shutter-press or remote-capture call, the camera takes 3 photos with a different exposure value for each photo. However, gphoto2 has very limited support for the Canon 1000D when it comes to the AEB feature. The exposure values for each HDR shot (-2eV, 0eV, +2eV) are to be set manually by the user in the camera's Settings menu (gphoto2 does not support setting this remotely when in AEB mode for the 1000D). Then the user needs to set the capture setting on the camera to **Burst Mode**. However, Canon 1000D cannot execute multiple photos in **Burst Mode**, either due to it being an unsupported feature or due to a bug. The only way to simulate this feature was to set the capture setting on the camera to **Timed Capture** (minimum available value is 2 seconds). The timer needs to be set so all three exposures can be taken consecutively, otherwise three different capture commands will need to be issued, which introduces further time overhead. Once this initial setup has been done by the user, the exposures for each HDR shot can be taken with a simple *(*gphoto2 –capture-image) command.

However, each call still takes 13 seconds to execute (a meagre saving of 2 seconds compared to Manual mode shooting). The 1000D can only take three exposures per AEB shot, and the exposure difference (in stops) cannot be adjusted using *gphoto* but must be done manually on the camera. Shooting in AEB mode is not suitable for scenes that need 5 or more exposures for optimal dynamic-range capture. Due to its limited support on the 1000D, this feature is unsuitable for use beyond simple scenes. In such simple scenes, shooting in AEB mode has the advantage that since *gphoto* only needs to be called once, the camera takes all three exposures one after the other, without any delay.

5.2.2 HDR Capture Parameters

The four HDR-timelapse related parameters are: number of exposures, exposure difference in stops, duration, interval. Number of exposures and exposure difference in stops determine the HDR image, duration and interval determine the number of frames available and the time between each HDR-capture for the final video. These parameters need to be set relative to each scene. For outdoor scenes involving illumination change e.g. sunrise or sunset, 2 f-stops need to be used to capture the most possible range and a higher number of exposures need to be captured for each HDR shot. This is because as the ISO, Aperture and Shutterspeed(s) will be the same constant values throughout the duration of exposure shooting, however in a sunset illumination will be continuously decreasing. Using exposure difference of 2 and at least 5 exposures means the lowest shutterspeed value will be 10 (5 * 2) stops from the highest, capturing a wide range of exposures and making up for the diminishing light in the scene.

The length of the final timelapse video can be calculated as thus:

$$timelapse\ video\ length = \frac{duration}{((time\ for\ one\ hdr\ shot) + interval) * frames\ per\ second} \tag{5.1}$$

Due to the tme it takes for gphoto2 to take multiple exposures(for each frame in final video), the time it takes to capture images needs to be factored in the equation as well.

If a timelapse capture is initiated from the Android App with the following parameters:
Duration = 60 minutes i.e. 3600 seconds
Exposures = 3
Interval = 0
Exposure difference = 1
During timelapse generation, the frames per second is set to 24.
the length of the output video when it is generated can be calculated according to Equation 5.1.

```
video length = 3600 / (15 + 0) * 24
```

For this example, video length will be 10 seconds. Note that this example assumes a low base shutterspeed (around 1/100), as this was the base shutterspeed used for calculating *time for one hdr shot*.

5.2.3 Merging exposures to HDR

Calibrating HDR Images

(a) Before Calibration

(b) After Calibration

Figure 5.6: Image Before/After Calibration. Calibrated by providing calibration matrix for Canon 1000D (see Section 4.2) to *pfscolortransform*

As part of thesis, a program called *pfscolortransform* was written (See Section 4.5). **This was contributed to existing open-source project *pfstools*.** An image before and after calibration is shown in Figure 5.6. Using the calibration matrix computed for the Canon 1000D camera, the calibrated image is desaturated in comparison to the original image. The sky is of a paler white than before.
Comparison of the HDR images after tonemapping is given in Appendix to compare them before and after calibration. Figure INSERT in Appendix shows HDR image after colour calibration and tonemapping with *pfstmo_mantiuk08* appears as the best image perceptually, when compared to the other tonemapped images before/after calibration.

Tonemapping HDR Images

Once the HDR images were created, to display them on the screen requires tone mapping them since the timelapse video uses JPG files which uses a limited set of colours. The unavailability of displays with sufficiently large dynamic range meant the HDR images had to be tone mapped to be displayed and to encode these into a timelapse video. Tone mapping ensures the full dynamic range is visually represented despite a Low Dynamic Range device being used to display it.

pfstools comes with a suite of tone map operators called pfstmo [67]. The following tone map operators, which are present in *pfstmo*, were tested to determine the operator which produces the most desirable results:

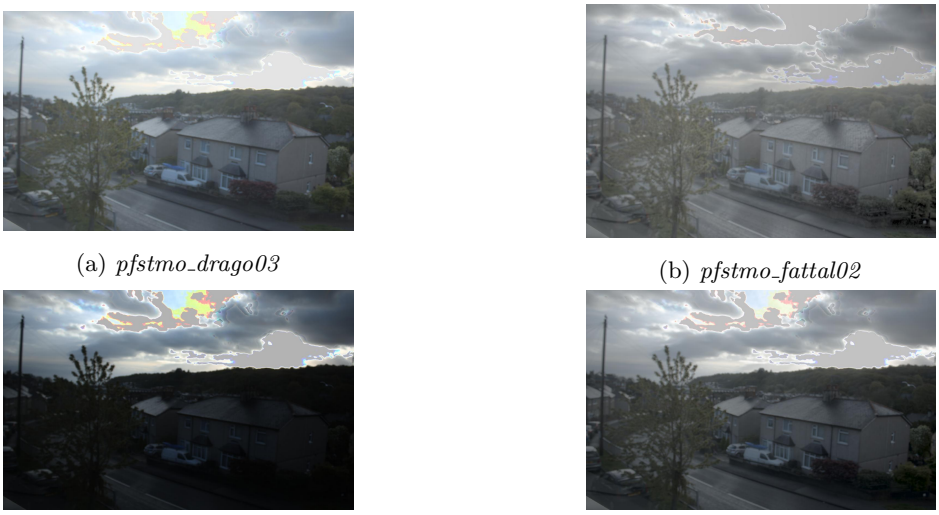
1. **pfstmo_drago03** [68]
2. **pfstmo_reinhard02** [69]
3. **pfstmo_fattal02** [70]
4. **pfstmo_mantiuk08** [71]

Figure 5.7: Fattal's tone map operator is not suitable for noisy images.

A tone mapped image was generated using the HDR files by applying each of the above tone map operators. The tone mapped images are shown in Figure INSERT. Each tone map operator has its strengths and weaknesses. **pfstmo_drago03** tonemap operator works on a pixel by pixel

basis and thus is faster than the other algorithms. This operator excels on noisy images, where other operators like **pfstmo_fattal02** produced bad images (Figure 5.7). **pfstmo_fattal02** amplifies noise on dark images, however for scenes with more illumination, this operator can produce surreal but non-realistic images. The images produced often contain bolder colours than in the original scene.

pfstmo_durand02 is known to produce the most realistic pictures, although it also needs a lot of time to compute the output [72].**pfstmo_reinhard02** and **pfstmo_reinhard04** are simple algorithms that have few processing steps. They were also used to test against the other tone mapping operators. **pfstmo_reinhard04** is also suitable for particularly problematic scenes like sunsets where towards the end of the capture duration, there is very low illumination and a lot of noise is captured.



(a) *pfstmo_drago03*

(b) *pfstmo_fattal02*

(c) *pfstmo_reinhard02*

(d) *pfstmo_mantiuk08*

Figure 5.8: colour-artifacts on images tonemapped with the four test tonemapping operators

Using *pfsview*, an HDR image viewing application included as part of the *pfstools* suite, a LDR image can be exported which maps the full dynamic range of the HDR image. This can be used to compare against the four tonemapped images in Figures **??** and 5.8. All four tonemapped images show **colour artefacts** [73] in the "clouds". This is very high in the Reinhard02 tonemapped image (Figure 5.8c) where the cloudy sky shows mostly bright yellow, red and cyan spots. For the yellow spots, it is probably as a result of the blue colour channel's values getting clipped, leaving red and green channels to form yellow.
Analysing the tonemapped images, *pfstmo_mantiuk08* produces the most visually appealing images, followed by *pfstmo_reinhard02*.

HDR Artifacts

Ghosting is introduced when moving objects across multiple exposures are merged into a single image [74]. Even though the exposures are shot consecutively, objects have moved in the scene from

(a) *pfstmo_drago03*

(b) *pfstmo_fattal02*

(c) *pfstmo_reinhard02*

(d) *pfstmo_mantiuk08*

Figure 5.9: Image of scene with Low illumination after tonemapping.

Figure 5.10: Ghosting of object (van)

one exposure to the next. During HDR processing when the exposures are merged, the object is duplicated in multiple positions as it is in different positions in each exposure. The object also looks faded and transparent, thus called ghosting. This can be seen in Figure 5.10, as the van in the image

shows ghosting (a faded transparent van is seen right behind the main van). Ghosting is an HDR artefact that can be reduced by using deghosting algorithms. One way in which these algorithms approach this problem is by using the optical flow approach: predict how every pixel moves from one exposure to the next and aligning exposures accordingly [73].
This can also be fixed manually using a image processing tool in post-production using software like Adobe Photoshop. This is a cumbersome process to apply to every HDR image and not viable for the bulk of HDR images created for each timelapse.

It must be noted that ghosting is different to motion blurring, which occurs within a single frame. Motion blurring happens when the shutterspeed chosen for the scene is too slow and an object moves between the time the camera opens and closes its shutter.

5.2.4 Timelapse Video

The mencoder program used to generate the timelapse video from the HDR files has a variety of parameters that can be changed. A noteworthy attribute to tweak however, is the FPS(frames per second) of the final video. 24 fps is the most widely used setting for film, but for scenes involving slow changes, the frame rate can be lowered until a desirable transition is achieved from one frame to the next. For example, to generate a timelapse video for slow moving clouds, a 12 fps achieved very perceptually desirable results. However, for scenes involving a lot of changes e.g. sunset timelapse-video, this can be detrimental as the video appears choppy and the transition between one frame to the next is not seamless.

Chapter 6

Conclusions & Future Work

6.1 Color Calibration

In the colour calibration aspect of this dissertation, we looked at calibrating three consumer cameras including one point-and-shoot camera. We took multiple exposures for each scene and camera and merged these exposures using the High Dynamic Ranging technique to form a single image with a wide dynamic range. This image captures more accurate information about bright and dark regions in the scene than traditional imaging techniques. We saw the results post calibration for all three scenes and how perceptually similar they were to each other, and contrasted this with the difference in camera images prior to calibration. In Section 4.2.1, we introduced the two models we used for calibration. Model A was found to be a good fit and a good predictor for finding the best transformation matrix. Model B was supposed to give us more freedom by capturing noise information but unexpectedly, this was not so and no improvements were found by using this complex model. We used the colour difference metric ΔE and also used relative difference in luminance since absolute difference in colours is not a good indicator especially for the human visual system.

We differed from previous studies by using RAW images which store linear intensity values and is the most accurate reproduction of the scene as captured by the camera sensor. This can be noted from the colour difference errors and luminance differences mentioned in the Tables, which are much lower than in the previous studies. We briefly mentioned the Just Noticeable Difference (JND) of 4.0, and how most colours in the testing scene fell below this threshold except for the complex scenes like "Conference Room" and "Library". Unlike previous studies, we have also visually represented the colours post-calibration so that they can be visually assessed. As noted above, except for one camera in one of the complex scenes, calibrated colours looked distinctly similar to the colours measured by the photospectrometer. Most of the colour differences were below the perceivable threshold. There are still some questions to be answered in this study which we will leave for future research. Finding negative values in the matrices implies presence of primaries which are outside the visible colour gamut, and this should be looked into. We would like to take into account factors like vignetting and glare, which have been shown in previous studies to cause small degrees of error in measurements. A number of the previous studies mentioned have taken optical vignetting into account, and this will

be something to consider in future work.

Given further opportunity to test this model, we would like to validate against scenes with a greater luminance variation. Previous studies like Anaokar et. al [38] have mentioned the high error rate associated with luminance measurements using HDR imaging in scenes with colours that have high chroma such as foliage. This would be something that should be tested in future work, as the scenes we used were building interiors. This could also help to test the presence of negative values and see which calibrated colours lie outside the sRGB colour gamut.

6.2 HDR Timelapse

The objectives set out at the beginning of the dissertation were not fully met. Colorimetric calibration was achieved (which was published in a conference) and using the calibration matrix, HDR images were calibrated. These were visually better than non-calibrated images, although the differences were barely noticeable. Software written as part of this dissertation was added to existing open-source software. Remote initiation of HDR exposures capture and the ability to preview current settings before starting a timelapse capture added a degree of automation to the system. The Raspberry Pi, used as a portable system, is adequate for HDR capture purposes. It is a suitable accessory along with the Android app and camera as a portable HDR capture system.
This system is not yet ready to be deployed publicly, and needs additional work and more testing before it is ready for such consideration.

6.2.1 Future Work

Due to time constraints, some features that would have considerable improved the system could not be implemented. The ability to cancel a timelapse which has been started from within the app would be a handy feature. Currently, cancelling exposure capture requires manually resetting the camera or disconnecting from the server's shell. Another feature that should be implemented in a future version should be the ability to preview an HDR image on the android app. One of the limitations of the current system is the unreliable nature of *gphoto2*. A good feature to implement for the app would be easy to understand error messages send to the app which will tell the user the specific error when a camera-capture call fails. Error messages should differentiate between "Camera not present", "Camera present, but cannot capture photo", "Error in shell script" etc. This will help the user to quickly debug the issue when an error arises.

The system was implemented for the Canon 1000D and tested exclusively on it, so more testing needs to be done to make sure this system works on other cameras. Two ways of exposure capture were discussed in this dissertation: Manual and Aperture-Priority modes. Aperture-Priority mode is the superior option, and performs well for challenging scenes like scenes with illumination changes during timelapse capture. One limiting factor is that the shutterspeed and aperture values were hard-coded into the Android App (in the drop-down menus). Future versions should dynamically load the shutterspeed values using:

```
gphoto2 --get-config=shutterspeed
```

```
gphoto2 --get-config=aperture
```

One way of implementing this could be to ask user to manually enter shutterspeeds from camera into the App at startup, or more suitably, send a command to the server requesting shutterspeed values which are then fed into the drop-down menus. Many DSLR cameras have Auto Exposure Bracketing (AEB) built in. As discussed in Section 5.2, there is only limited support for AEB on the camera used for this dissertation, thus it was not a viable option. However, cameras for which *gphoto2* supports AEB control should be tested to see if using AEB can improve the capturing process.

The hardware limitations of the Raspberry Pi have previously been discussed. In future, a more powerful single-board computer could be used to see if the processes of exposure capture, merging and timelapse video-generation can be developed into a cohesive system running on a single hardware unit. This has a array of benefits, as the App can essentially launch the merging and video-generation process. A possible extension could be that the user can download the generated timelapse video onto the phone once it is generated.

Chapter 7

Legal, Social, Ethical and Professional Issues

No copyrighted materials were required to be used for the dissertation. All the images used for testing and training were user-owned.

The software used in this dissertation and their associated licensing is given below:

gphoto2 is freely available and distributed under the terms of the GNU General Public Licence.

pfstools is also free software; it comes under the terms of the GNU General Public License.

pfscolortransform, created as part of this dissertation, was added to **pfstools** and will be published under the terms of the GNU General Public License.

The Android Application, **RemoteTimelapse** is in closed beta, and has not been made publicly available.

Bibliography

[1] D Varghese, R Wanat, and R.K. Mantiuk. Colorimetric calibration of high dynamic range images with a ColorChecker chart. In *HDRi2014 - Second International Conference and SME Workshop on HDR imaging (2014)*, pages 17–22, 2014.

[2] JETI Technische Instrumente GmbH. specbos 1211. http://www.jeti.com/cms/index.php/instruments-55/radiometer/specbos-1211. [Online; accessed 27-March-2014].

[3] Jwrodgers. RaspberryPi Image. http://commons.wikimedia.org/wiki/File:RaspberryPi.jpg, 2012.

[4] Ryan Chylinski. What is Time-lapse Photography? http://www.learntimelapse.com/time-lapse-photography-how-to-guide/what-is-time-lapse-photography/, 2012.

[5] Tom Lowe. Time Lapse - TimeScapes.org. http://www.timescapes.org/time-lapse/, 2013.

[6] Dan jacob. Time Lapse Blooming of a Geranium. http://www.youtube.com/watch?v=fqOB3atVjOw, 2007.

[7] dnp denmark. Luminance is the only form of light we can see. http://www.dnp-screens.com/DNP08/Technology/Basic-Visual/What-is-light/Luminance.aspx.

[8] schorsch.com. Luminance, Photometric Brightness. http://www.schorsch.com/en/kbase/glossary/luminance.html, 2004.

[9] HDRsoft. HDR images in photography - About Dynamic Range, Tone Mapping and HDR Imaging for Photography. http://www.hdrsoft.com/resources/dri.html.

[10] Cambridge in Colour. DIGITAL CAMERA SENSORS. http://www.cambridgeincolour.com/tutorials/camera-sensors.htm.

[11] Cambridge in Colour. DYNAMIC RANGE IN DIGITAL PHOTOGRAPHY. http://www.cambridgeincolour.com/tutorials/dynamic-range.htm.

[12] Darren Rowse. Auto Exposure bracketing (AEB) - Digital Photography School. http://digital-photography-school.com/automatic-exposure-bracketing-aeb/.

[13] hdr photography. Auto Exposure bracketing. http://www.hdr-photography.com/aeb.html.

[14] Cambridge in Colour. Tutorials: The RAW file format. http://www.cambridgeincolour.com/tutorials/RAW-file-format.htm.

[15] Darren Rowse. RAW vs JPEG - Digital Photography School. http://digital-photography-school.com/raw-vs-jpeg/.

[16] Michael Reichmann. Understanding Raw Files. http://www.luminous-landscape.com/tutorials/understanding-series/u-raw-files.shtml.

[17] Cambridge in Colour. Image Types: JPEG and TIFF Files. http://www.cambridgeincolour.com/tutorials/imagetypes.htm.

[18] Kresimir Matkovic. Colorimetry. http://www.cg.tuwien.ac.at/research/theses/matkovic/node14.html.

[19] photonics.com. Coloriemtry, photonics handbook. http://www.photonics.com/EDU/Handbook.aspx?AID=25124.

[20] Mark Fairchild. *Color Appearance Models (Second Edition).* Wiley, 1997.

[21] David Madore. Colors and colorimetry. http://www.madore.org/~david/misc/color/#cie.

[22] Charles A. Poynton. *Digital Video and HDTV: Algorithms and Interfaces.* Morgan Kaufmann, 2003.

[23] gamma sci.com. What is colorimetry? http://www.gamma-sci.com/colorimetry/.

[24] R. W. G Hunt. *The Reproduction of Colour (6th ed.).* Chichester UK: WileyIS&T, 2004.

[25] Adobe Technical Guides. The rgb, cmy color models. http://dba.med.sc.edu/price/irf/Adobe_tg/models/rgbcmy.html.

[26] Adobe Technical Guides. Ciexyz color models. http://dba.med.sc.edu/price/irf/Adobe_tg/models/ciexyz.html.

[27] 3nh.com. Cie standard observers: 3nh. http://www.3nh.com/en/news/en_64.html.

[28] Sareesh Sudhakaran. What is the difference between cie lab, cie rgb, cie xyy and cie xyz? http://wolfcrow.com/blog/what-is-the-difference-between-cie-lab-cie-rgb-cie-xyy-and-cie-xyz/.

[29] Douglas Kerr. The cie xyz and xyy color spaces. http://graphics.stanford.edu/courses/cs148-10-summer/docs/2010--kerr--cie_xyz.pdf.

[30] Randy klimek. What is the difference between radiometers, spectrometers, and spectroradiometers? http://sensing.konicaminolta.us/2013/11/what-is-the-difference-between-radiometers-spectrometers-and-spectroradiometers/.

[31] Bruce Lindbloom. xyy to xyz. http://www.brucelindbloom.com/index.html?Eqn_xyY_to_XYZ.html.

[32] Bruce Lindbloom. Delta e (cie 1994). http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE94.html.

[33] Steve Upton. Deltae the color difference. http://www.colorwiki.com/wiki/Delta_E:_The_Color_Difference.

[34] Bruce Lindbloom. Delta e (cie 2000). http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE2000.html.

[35] Martin Habekost. Which color differencing equation should be used? http://www.hdm-stuttgart.de/international_circle/circular/issues/13_01/ICJ_06_2013_02_069.pdf.

[36] Gaurav Sharma, Wencheng Wu, and Edul Dalal. The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. Xerox Corporation, 800 Phillips Road, Webster, NY 14580, 2004.

[37] MN Inanici. Evaluation of high dynamic range photography as a luminance data acquisition system. *Lighting Research and Technology*, 38(2):123–134, 2006.

[38] Smita Anaokar and Martin Moeck. Validation of high dynamic range imaging to luminance measurement. *Leukos*, 2(2):133–144, 2005.

[39] Martin Moeck. Accuracy of luminance maps obtained from high dynamic range images. *Leukos*, 4(2):99–112, 2007.

[40] Grzegorz Krawczyk, Michael Goesele, and Hans-Peter Seidel. Photometric calibration of high dynamic range cameras. Technical report, Max-Planck-Institut für Informatik, 2005.

[41] Yulia I Tyukhova. The assessment of high dynamic range luminance measurements with led lighting. Master's thesis, University of Nebraska, 2012.

[42] Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Color calibrated high dynamic range imaging with icc profiles. In *Color and Imaging Conference*, volume 2001, pages 286–290. Society for Imaging Science and Technology, 2001.

[43] Chainfire. DSLR Controller Android App. https://play.google.com/store/apps/details?id=eu.chainfire.dslrcontroller, 2013. latest version v0.99.4 BETA.

[44] ZigMaj. DSLR Remote Controller Android App. https://play.google.com/store/apps/details?id=us.zig.dslr&hl=en, 2011.

[45] Geeking About. Controlling a Nikon D5200 with Raspberry Pi and gPhoto2 2.5.2. http://www.geekingabout.com/controlling-a-nikon-d5200-with-raspberry-pi-and-gphoto2-2-5-2.html, 2013.

[46] Khurt Williams. HDR photography with Raspberry Pi and gPhoto2. http://islandinthenet.com/2012/08/hdr-photography-with-raspberry-pi-and-gphoto2/, 2012.

[47] Moreno Marzolla. Time-Lapse movies with gphoto2. http://www.moreno.marzolla.name/software/linux-time-lapse/, 2013.

[48] fotosyn. Simple timelapse camera using Raspberry Pi and a coffee tin. http://www.instructables.com/id/Simple-timelapse-camera-using-Raspberry-Pi-and-a-c/, 2013.

[49] James Bruce. How To Capture Time-Lapse Photography With Your Raspberry Pi and DSLR or USB Webcam. http://www.makeuseof.com/tag/how-to-capture-time-lapse-photography-with-your-raspberry-pi-and-dslr-or-usb-webcam/, 2013.

[50] Andrew Back. Time-lapse Photography with the Raspberry Pi Camera. http://www.designspark.com/blog/time-lapse-photography-with-the-raspberry-pi-camera, 2013.

[51] Trevor Appleton. Creating a Time-Lapse Camera with the Raspberry Pi and Python. http://trevorappleton.blogspot.co.uk/2013/11/creating-time-lapse-camera-with.html, 2013.

[52] H.E. Ross and D. J. Murray. *E.H.Weber on the tactile senses. 2nd ed.* Psychology Press, 1996.

[53] Centre for Intelligent Machines (CIM). Weber-fechner law. http://www.cim.mcgill.edu/~image529/TA529/Image529_99/projects98/30_Colm_Elliott/weber.html. [Online; accessed 13-March-2014].

[54] Konica Minolta. Features of cie 2000 color difference formula. http://www.konicaminolta.com/instruments/knowledge/color/part5/03.html. [Online; accessed 13-March-2014].

[55] Bruce Fraser, Chris Murphy, and Fred Bunting. *Real World Color Management (2nd Edition).* Peachpit Press, 2004.

[56] Rafal Mantiuk, Grzegorz Krawczyk, Radoslaw Mantiuk, and Hans-peter Seidel. High dynamic range imaging pipeline: perception-motivated representation of visual content. In *Human Vision and Electronic Imaging*, pages 649212–12, 2007.

[57] Dave Coffin. DCRAW software, 2013, version 9.20. http://www.cybercom.net/ dcoffin/dcraw/.

[58] Scott Delap. Google's Android SDK Bypasses Java ME in Favor of Java Lite and Apache Harmony. http://www.infoq.com/news/2007/11/android-java, 2007.

[59] Dan Bornstein. Dalvik Technical Information. https://source.android.com/devices/tech/dalvik/index.html, 2008.

[60] et al. Fielding. RFC 2616. http://tools.ietf.org/html/rfc2616#section-9.5, 1999.

[61] Raspberry Pi Foundation. FAQs — Raspberry Pi. http://www.raspberrypi.org/help/faqs/, 2014.

[62] Roderick W. Smith. *Advanced Linux Networking.* Addison-Wesley Professional, 2002.

[63] Stewart Watkiss. Running a lightweight webserver on the Raspberry Pi (lighttpd). http://www.penguintutor.com/linux/light-webserver, 2012.

[64] R Mantiuk, K Myszkowski, and H.P. Seidel. A Perceptual Framework for Contrast Processing of High Dynamic Range Images. In *In ACM Transactions on Applied Perception, 2006*, 2006.

[65] Photo tip can your camera see infrared? http://kentweakley.com/blog/photo-tip-camera-infrared/. [Online; accessed 02-May-2014].

[66] Darren Rowse. Aperture and Shutter Priority Modes. http://digital-photography-school.com/aperture-and-shutter-priority-modes/, 2013.

[67] Grzegorz Krawczyk. pfstmo::implementation of tone mapping operators. http://www.mpi-inf.mpg.de/resources/tmo/, 2007.

[68] F Drago, K Myszkowski, T Annen, and N Chiba. Adaptive Logarithmic Mapping for Displaying High Contrast Scenes. In *In Eurographics 2003*, 2003.

[69] E Reinhard, M Stark, P Shirley, and J Ferwerda. Photographic Tone Reproduction for Digital Images. In *In ACM Transactions on Graphics, 2002*, 2002.

[70] R Fattal, D Lischinski, and M Werman. Gradient Domain High Dynamic Range Compression. In *In ACM Transactions on Graphics, 2002*, 2002.

[71] Rafal Mantiuk, Scott Daly, and Louis Kerofsky. Display Adaptive Tone Mapping. In *In: ACM Transactions on Graphics 27 (3), 2008.*, 2008.

[72] Open Source Photography. Parameters for tone mapping operators. http://osp.wikidot.com/parameters-for-photographers, 2011.

[73] K Karaduzovic-Hadziabdic, J Telalovic, and R Mantiuk. Expert evaluation of deghosting algorithms for multi-exposure high dynamic range imaging. In *HDRi2014 - Second International Conference and SME Workshop on HDR imaging (2014)*, pages 12–16, 2014.

[74] Pye Jirsa. Motion Blur vs Ghosting: The Difference between these 2 artifacts. http://www.slrlounge.com/motion-blur-vs-ghosting-the-difference-between-these-2-artifacts, 2013.

Appendix

Refer to **User Guide** in attached CD-ROM for software used in this dissertation and deployment instructions.

CIE xyY readings from ColorChecker chart, taken using spectroradiometer, used for colour calibration

```
36.44,0.4203,0.3845
111.8,0.4241,0.3786
58.5,0.2798,0.296
47.66,0.3596,0.4712
76.4,0.3039,0.2876
135.5,0.2915,0.4009
112.4,0.5198,0.4162
36.18,0.2412,0.2027
71.15,0.4951,0.3343
21.78,0.3216,0.2467
159.1,0.3853,0.5213
159.4,0.4724,0.4668
19.51,0.2173,0.1651
88.18,0.3189,0.5293
47.2,0.5472,0.35
215.8,0.4511,0.4969
66.29,0.4101,0.2652
60.16,0.2334,0.2999
305.9,0.3474,0.3657
200.7,0.3451,0.3624
125.7,0.3426,0.3616
65.4,0.3452,0.363
31.76,0.3411,0.3606
11.86,0.342,0.3598}
```

RGB Readings of ColorChecker chart HDR using Canon 1000D

```
1146.3,751.92,519.58
3698.3,2354.1,1760
1195.5,1701.7,2234.5
949.25,1166.1,520.65
1852.7,2059.6,2808.7
2195.6,3750.4,3021
4397.7,1767.3,572.55
772.73,1209.7,2545.5
3275.3,1102.9,975.85
699.79,554.81,964
3180.8,3684.9,1044.2
4763.3,2951.7,729.88
378.74,720.81,1804.6
1278.9,2340.5,886.17
2393.9,643.68,408.4
5773.3,4402.3,916.12
2884.5,1232.7,2016.4
826.65,1999.9,2604.6
7283.6,7295.6,6492
4878.6,4931.1,4484.6
3022.6,3073,2801.3
1582.6,1610.6,1452.1
755.55,791.59,725.56
294.48,304,278.13
```

Calibrated vs uncalibrated HDR image after tonemapping

(a) Before Calibration

(b) After Calibration

Figure 7.1: Image Before/After Calibration. Tonemapped with *pfstmo_drago03*

(a) Before Calibration

(b) After Calibration

Figure 7.2: Image Before/After Calibration. Tonemapped with *pfstmo_fattal02*

(a) Before Calibration

(b) After Calibration

Figure 7.3: Image Before/After Calibration. Tonemapped with *pfstmo_reinhard02*

(a) Before Calibration

(b) After Calibration

Figure 7.4: Image Before/After Calibration. Tonemapped with *pfstmo_mantiuk08*

Colour Calibration

Extract RGB Values from HDR file (MATLAB)

```
numValues=7;
patchSize=40;
% FileName='550d_cc1.hdr';
```

```
tic;
patchPixels = ceil(patchSize/2);
[FileName,PathName] = uigetfile('*.hdr','Select the HDR file');
fullpath = strcat(PathName,FileName);
% fullpath='/home/delvin/Dropbox/Public/Colorimetry/HDRs/HDR/550d_cc1.hdr';
hdr = pfs_read_image(fullpath);
f=figure;
imshow(hdr/(0.05*max(hdr(:))));

rgbV = zeros(numValues,3);
[x,y]=ginput(numValues);
x=floor(x);
y=floor(y);
% 24 x 2 x&y position!
for i=1:numValues
    rgbPatch= hdr((y(i)-patchPixels):(y(i)+patchPixels),
                    (x(i)-patchPixels):(x(i)+patchPixels),:);
    rgbV(i,:) = [mean2(rgbPatch(:,:,1)) mean2(rgbPatch(:,:,2)) mean2(rgbPatch(:,:,3))];
end
fil = strcat('RGBv_',FileName,'.csv');
csvwrite(fil,rgbV);
close all;
TimeTaken=toc
```

Compute Calibration Matrix using spectroradiometer xyY readings and RGB readings (MATLAB)

```
[FileName,PathName] = uigetfile('*.csv','Select the RGB file');
rgb_file=csvread(strcat(PathName,FileName));

[File2,Path2] = uigetfile('*.csv','Select the Yxy file');
yxy_file=csvread(strcat(Path2,File2));

for i=1:size(yxy_file,1)
    xyz_data(i,1) = yxy_file(i,2) * (yxy_file(i,1)/yxy_file(i,3));
    xyz_data(i,2) = yxy_file(i,1);
    xyz_data(i,3) = (1 - yxy_file(i,2) - yxy_file(i,3)) *
                        (yxy_file(i,1)/yxy_file(i,3));
end
r2x = rgb_file \ xyz_data
```

Server PHP Scripts

preview.php

```
<?php
if(isset($GLOBALS['HTTP_RAW_POST_DATA']) == TRUE){
        echo $GLOBALS['HTTP_RAW_POST_DATA'];
    }else{
        echo $HTTP_RAW_POST_DATA;
    }
$iso=100;
$shutterspeed=1/250;
$aperture=5;

$iso = $_POST['iso'];
$shutterspeed = $_POST['shutterspeed'];
$aperture = $_POST['aperture'];

//$url = exec("sudo ./gPreview 1/100 2>&1", $out, $err);
//print_r($out);
system("rm -f camprev.JPG");
$out=system("sudo -u pi /home/pi/scripts/gphoto2_reset.sh
            --set-config shutterspeed=$shutterspeed
            --set-config iso=$iso
            --set-config aperture=$aperture
            --set-config imageformat=5
            --filename 'camprev.JPG'
            --capture-image-and-download");
echo "ISO is $iso, ss is $shutterspeed and aperture is $aperture";
//echo "Error: $err ";

?>
```

timelapse.php

```
<?php

if(isset($GLOBALS['HTTP_RAW_POST_DATA']) == TRUE){
        echo $GLOBALS['HTTP_RAW_POST_DATA'];
    }else{
        echo $HTTP_RAW_POST_DATA;
```

```
    }

// Initialise Values
$iso=100;
$shutterspeed="1/250";
$aperture=5;
$duration=300;
$exposures=3;
$interval=0;
$fstops=1;
// Retrieve Capture parameters from POST

$iso = $_POST['iso'];
$aperture = $_POST['aperture'];
$shutterspeed = $_POST['shutterspeed'];

$duration = $_POST['duration'];
$exposures = $_POST['exposures'];
$interval = $_POST['interval'];
$fstops = $_POST['fstops'];

// Start timelapse with required number of f-stops
if($fstops!=1){
    $out=system("sudo -u pi /home/pi/scripts/timelapse.sh
        -i $iso -s $shutterspeed -a $aperture -d $duration
        -e $exposures -l $interval -f");
}
else
{
    $out=system("sudo -u pi /home/pi/scripts/timelapse.sh
        -i $iso -s $shutterspeed -a $aperture -d $duration
        -e $exposures -l $interval");
}
$out1="banana hammock";
echo $out1;

system("rm -f Output.log");
$fileWrite = "Output.log";
$fh = fopen($fileWrite, 'w');
fwrite($fh, $out);
fwrite($fh, $duration);
```

```
fwrite($fh, "\nAND\n");
fwrite($fh, $shutterspeed);
fclose($fh);

/*
$combi = $duration . $exposures;

// echo "Hello Dumie Delvin $combi, $duration, $aperture, $shutterspeed";
$output = shell_exec('dir');
echo "$output";

$fileWrite = "dumiieFile-$combi.txt";
$fh = fopen($fileWrite, 'w');

fwrite($fh, "Duration is $duration minutes, while Aperture is $aperture.\n
Shutterspeed is $shutterspeed seconds.");
fclose($fh);
*/
?>
```

Timelapse video scripts

gphoto2_reset.sh

```
#!/bin/bash
#
dev=`gphoto2 --auto-detect | grep usb | cut -b 36-42 | sed 's/,/\//'`
if [ -z ${dev} ]
then
   echo "Error: Camera not found"
   exit 255
fi
sudo /home/pi/scripts/usbreset /dev/bus/usb/${dev} > /dev/null
gphoto2 $@
rc=$?
sudo /home/pi/scripts/usbreset /dev/bus/usb/${dev} > /dev/null
exit $rc
```

timelapse.sh

```
#!/bin/bash
```

```
#Camera Parameters
iso=100
aperture=5
baseshutter=1/250
baseshutterIndex=40
# HDS parameters
fstops=1
exposures=3
interval=0
duration=200

# Manual mode or Av mode
cameraMode=M
#Searches array passed as arg1 for arg2
getIndex() {
    for ((index=0; index<${#shutterspList[@]}; index++)); do
        if [ "${shutterspList[$index]}" = "$baseshutter" ]; then
            echo "ShutterIndex is $index"
            return
        fi
    done
    index=-1
    echo 'Invalid ShutterSpeed value entered.
    Exiting....'
    exit -1
}

#Setup capture parameters here
initialSetup() {
    gphoto2_reset.sh --set-config imageformat=7
        --set-config iso="$iso"
        --set-config aperture="$aperture"
}

#Shoot HDR exposures here
shootHDR() {
    for ((i=0; i<$exposures; i++ )); do
        gphoto2_reset.sh --set-config
            shutterspeed="${shootExposures[$i]}"
            --capture-image
```

```
    done
}

shootAvHDR(){
    if [ "$exposures" == "3" ]; then
        if [ "$fstops" = "2" ]; then
            gphoto2_reset.sh --set-config exposurecompensation="0"
                --capture-image
            gphoto2_reset.sh --set-config exposurecompensation="6"
                --capture-image
            gphoto2_reset.sh --set-config exposurecompensation="12"
                --capture-image
        else
            gphoto2_reset.sh --set-config exposurecompensation="3"
                --capture-image
            gphoto2_reset.sh --set-config exposurecompensation="6"
                --capture-image
            gphoto2_reset.sh --set-config exposurecompensation="9"
                --capture-image
        fi
    elif [ "$exposures" == "5" ]; then
        gphoto2_reset.sh --set-config exposurecompensation="0"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="3"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="6"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="9"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="12"
             --capture-image
    elif [ "$exposures" == "7" ]; then
        gphoto2_reset.sh --set-config exposurecompensation="0"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="2"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="4"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="6"
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="8"
```

```
            --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="10"
             --capture-image
        gphoto2_reset.sh --set-config exposurecompensation="12"
             --capture-image
    fi
}
#Print usage message
function usage
{
    echo "usage: system_page [[[-f file ] [-i]] | [-h]]"
}


##### Main

while [ "$1" != "" ]; do
    case $1 in
        -a | --aperture )       shift
                                aperture=$1
                                ;;
        -d | --duration )       shift
                                duration=$1
                                ;;
        -e | --exposures )      shift
                                exposures=$1
                                ;;
        -f | --fstops )         fstops=2
                                ;;
        -h | --help )           usage
                                exit
                                ;;
        -i | --iso )            shift
                                iso=$1
                                ;;
        -l | --interval )       shift
                                interval=$1
                                ;;
        -s | --shutterspeed )   shift
                                baseshutter=$1
                                ;;
```

```
        * )                     usage
                                exit 1
    esac
    shift
done

# Check if camera is connected and in Manual or Aperture-Priority mode
if gphoto2 --get-config autoexposuremode | grep -q "Current: Manual"; then
    cameraMode=M
else
    if gphoto2 --get-config autoexposuremode | grep -q "Current: AV"; then
        cameraMode=AV
    else
        echo "Camera needs to be connected and in Manual or Av mode."
        exit
    fi
fi


# Test code to verify command line processing

if [ "$fstops" = "2" ]; then
        echo "Using 2 f-stops..."
fi
echo "ISO = $iso
Aperture = $aperture
Shutter Speed = $baseshutter
Exposures = $exposures
Interval = $interval
Duration = $duration"

shutterspList=("30" "25" "20" "15" "13" "10" "8" "6" "5" "4" "3.2" "2.5"
    "2" "1.6" "1.3" "1" "0.8" "0.6" "0.5" "0.4" "0.3" "1/4" "1/5" "1/6"
    "1/8" "1/10" "1/13" "1/15" "1/20" "1/25" "1/30" "1/40" "1/50" "1/60"
    "1/80" "1/100" "1/125" "1/160" "1/200" "1/250" "1/320" "1/400"
    "1/500" "1/640" "1/800" "1/1000")
# Find index of base shutter speed if provided by user
getIndex


# Calculate List of exposures to shoot for one HDR-shot
```

```
if [ "$exposures" == "3" ]; then
    shootExposures=("${shutterspList[$index]}"
        "${shutterspList[$index-$fstops]}"
        "${shutterspList[$index+$fstops]}")
elif [ "$exposures" == "5" ]; then
    shootExposures=("${shutterspList[$index]}"
        "${shutterspList[$index-$fstops-$fstops]}"
        "${shutterspList[$index-$fstops]}"
        "${shutterspList[$index+$fstops]}"
        "${shutterspList[$index+$fstops+$fstops]}")
elif [ "$exposures" == "7" ]; then
    shootExposures=("${shutterspList[$index]}"
        "${shutterspList[$index-$fstops-$fstops-$fstops]}"
        "${shutterspList[$index-$fstops-$fstops]}"
        "${shutterspList[$index-$fstops]}"
        "${shutterspList[$index+$fstops]}"
        "${shutterspList[$index+$fstops+$fstops]}"
        "${shutterspList[$index+$fstops+$fstops+$fstops]}")
fi

# Run initial setup of Camera
initialSetup

# Call function to shoot HDRs.
# Duration is given in minutes.
timeleft=$duration
DIFF=0
START=$(date +%s)
while (( $DIFF < $duration )); do
    if [ "$cameraMode" == "M" ]; then
        echo "Shooting in M mode"
        shootHDR
    else
        echo "Shooting in Av mode"
        shootAvHDR
    fi
    echo "Interval of $interval seconds..."
    sleep $interval
    END=$(date +%s)
    DIFF=$(echo "$END - $START" | bc)
    timeleft=$(echo "$duration - $DIFF" | bc)
```

```
    echo "$DIFF seconds elapsed. $timeleft left out of $duration."
done
```

hdrgenerator.sh

```
#!/bin/bash

exposures=3 # Default 3 exposures unless specified by user

#Print usage message
function usage
{
    echo "usage: hdrgenerator.sh [[[-e exposures ] [-d download ]
        [-m mergetonemap ] [-c createmovie ] ]"
}

downloadImages() {
mkdir -p camera_photos # Create folder if doesnt exist
cd camera_photos # Change to the folder.

gphoto2_reset.sh -P  # Download all photos from camera to folder above.
if [ "$?" != "0" ];then
exit 1 # Exit if Download didnt succeed.
fi
}


mergeExposures() {
rawFiles=(*.CR2)
lengthArr="${#rawFiles[@]}"
indx=0
for (( i=0; i<$lengthArr; i=i+exposures )); do
leadDigits="$(printf "%03d" $indx)"
if [ "$exposures" == "3" ]; then
echo "Processing HDR $leadDigits : ${rawFiles[$i]}
                ${rawFiles[$i+1]} ${rawFiles[$i+2]}"
pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}"
                "${rawFiles[$i+2]}" | pfshdrcalibrate -r linear
                -v --bpp 16 | pfssize -x 1280 -y 800
                | pfsout --fix-halfmax Merge"$leadDigits".exr
elif [ "$exposures" == "5" ]; then
```

```
echo "Processing HDR $leadDigits :
                ${rawFiles[$i]} ${rawFiles[$i+1]} ${rawFiles[$i+2]}
                ${rawFiles[$i+3]} ${rawFiles[$i+4]}"
 pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}"
                "${rawFiles[$i+2]}" "${rawFiles[$i+3]}"
                "${rawFiles[$i+4]}" | pfshdrcalibrate -r linear
                -v --bpp 16 | pfssize -x 1280 -y 800 |
                pfsout --fix-halfmax Merge"$leadDigits".exr
elif [ "$exposures" == "7" ]; then
echo "Processing HDR $leadDigits : ${rawFiles[$i]} ${rawFiles[$i+1]}
                ${rawFiles[$i+2]} ${rawFiles[$i+3]} ${rawFiles[$i+4]}
                ${rawFiles[$i+5]} ${rawFiles[$i+6]}"
pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}" "${rawFiles[$i+2]}"
                "${rawFiles[$i+3]}" "${rawFiles[$i+4]}" "${rawFiles[$i+5]}"
                "${rawFiles[$i+6]}" | pfshdrcalibrate -r linear -v --bpp
                16 | pfssize -x 1280 -y 800 | pfsout
                --fix-halfmax Merge"$leadDigits".exr
fi
((indx++))
done
}

tonemapImage() {
exrFiles=(*.exr)
lenExrArr="${#exrFiles[@]}"
indx=0
for (( k=0; k<$lenExrArr; k++ )); do
leadDigits="$(printf "%03d" $indx)"
pfsin "${exrFiles[$k]}" | pfscolortransform -x 1000d_calibration.matrix |
            pfstmo_mantiuk08 | pfsout mantiukCalibratedHdr"$leadDigits".jpg
((indx++))
done
}

mergeAndTonemapImage() {
rawFiles=(*.CR2)
lengthArr="${#rawFiles[@]}"
indx=0
for (( i=0; i<$lengthArr; i=i+exposures )); do
leadDigits="$(printf "%03d" $indx)"
if [ "$exposures" == "3" ]; then
```

```
echo "Processing HDR $leadDigits : ${rawFiles[$i]}
                ${rawFiles[$i+1]} ${rawFiles[$i+2]}"
pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}"
                "${rawFiles[$i+2]}" | pfshdrcalibrate -r linear
                -v | pfssize -x 1280 -y 800 | pfstmo_drago03 |
                pfsgamma -g 1.7 | pfsout Hdr"$leadDigits".jpg
elif [ "$exposures" == "5" ]; then
echo "Processing HDR $leadDigits : ${rawFiles[$i]}
                ${rawFiles[$i+1]} ${rawFiles[$i+2]}
                ${rawFiles[$i+3]} ${rawFiles[$i+4]}"
pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}"
                "${rawFiles[$i+2]}" "${rawFiles[$i+3]}"
                "${rawFiles[$i+4]}" | pfshdrcalibrate -r linear
                -v | pfssize -x 1280 -y 800 | pfstmo_drago03 |
                pfsgamma -g 1.7 | pfsout Hdr"$leadDigits".jpg
elif [ "$exposures" == "7" ]; then
echo "Processing HDR $leadDigits : ${rawFiles[$i]}
                ${rawFiles[$i+1]} ${rawFiles[$i+2]}
                ${rawFiles[$i+3]} ${rawFiles[$i+4]}
                ${rawFiles[$i+5]} ${rawFiles[$i+6]}"
pfsinme "${rawFiles[$i]}" "${rawFiles[$i+1]}"
                "${rawFiles[$i+2]}" "${rawFiles[$i+3]}"
                "${rawFiles[$i+4]}" "${rawFiles[$i+5]}"
                "${rawFiles[$i+6]}" | pfshdrcalibrate -r linear
                -v | pfssize -x 1280 -y 800 | pfstmo_drago03 |
                pfsgamma -g 1.7 | pfsout Hdr"$leadDigits".jpg
fi
((indx++))
done
}

createMovie() {
mencoder  mf://*.jpg  -mf fps=20:type=jpg -oac copy
        -ovc xvid -xvidencopts fixed_quant=1:chroma_opt:
        vhq=4:max_bframes=1:quant_type=mpeg:threads=8
        -o output.mp4
}
##### Main
DIFF=0
START=$(date +%s)
```

```
while [ "$1" != "" ]; do
    case $1 in
        -e | --exposures )      shift
                                exposures=$1
                                ;;
        -h | --help )           usage
                                exit
                                ;;
        -d | --download )       downloadImages
                                ;;
        -j | --justmerge )      mergeExposures
                                ;;
        -t | --tonemap )       tonemapImage
                                ;;
        -m | --mergetonemap )   mergeAndTonemapImage
                                ;;
        -c | --createmovie )    createMovie
                                ;;
        * )                     usage
                                exit 1
    esac
    shift
done

END=$(date +%s)
DIFF=$(echo "$END - $START" | bc)
echo "$DIFF seconds elapsed."
```

Colour calibration of HDR file

pfscolortransform

```
/**
 * @brief Apply color transformation on the pfs stream
 *
 * This file is a part of PFSTOOLS package.
 * ---------------------------------------------------------------------
 * Copyright (C) 2003,2004 Rafal Mantiuk and Grzegorz Krawczyk
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
```

```
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 * ---------------------------------------------------------------------
 *
 * @author Delvin Varghese, <delvin.friends@gmail.com>
 *
 */

#include <config.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <iostream>

#include <getopt.h>
#include <pfs.h>

#define NUMELEMS(x)  (sizeof(x)/sizeof(*(x)))
#define PROG_NAME "pfscolortransform"

/*  EXAMPLE CONTENTS OF matrix_file.txt
  0.111424,0.222579,0.333464
  0.111656,0.222158,0.333186
  0.111332,0.222193,0.333444
*/
// static const float rgb2xyzD65Mat[3][3] =
static const float rgb2xyzD65Mat[3][3] =
{ { 0.412424f, 0.357579f, 0.180464f },
  { 0.212656f, 0.715158f, 0.072186f },
  { 0.019332f, 0.119193f, 0.950444f } };
```

```
// static const float xyz2rgbD65Mat[3][3] =
static const float xyz2rgbD65Mat[3][3] =
{ {  3.240708, -1.537259, -0.498570 },
  { -0.969257,  1.875995,  0.041555 },
  {  0.055636, -0.203996,  1.057069 } };

class QuietException {};

void printHelp()
{
  std::cerr << PROG_NAME " [--xyzrgb matrix_file.txt | --rgbxyz matrix_file.txt] [--transpose]" <<
            << "See man page for more information." << std::endl;
}

float** initialiseArray(const float referenceMat[][3]){
  float** outputMat;
  outputMat = new float*[3];
  for (int i = 0;i<3;i++){
        outputMat[i] = new float[3];
    }
    for (int i=0;i<3;i++){
        for(int j=0;j<3;j++){
          outputMat[i][j] = referenceMat[i][j];
        }
      }
    return outputMat;
}

float** readMatrixFile(char* fn, bool transpose){
  float** matArray;
    FILE *file = fopen(fn, "r");
    matArray = new float*[3];
    if ( file ){
        for (int i = 0;i<3;i++){
            matArray[i] = new float[3];
        }
        size_t i, j, k;
        char buffer[BUFSIZ], *ptr;
        for ( i = 0; fgets(buffer, sizeof buffer, file); ++i ){
            for ( j = 0, ptr = buffer; j <= NUMELEMS(*matArray); ++j, ++ptr ){
```

```
                if(!transpose){
                    matArray[i][j] = (float)strtof(ptr, &ptr);
                } else {
                    matArray[j][i] = (float)strtof(ptr, &ptr);
                }
            }
        }
        fclose(file);

        fprintf( stderr, "File Closed....\n");
    }
    else{
        fprintf( stderr, "ERROR \t ERROR");
        perror(fn);
    }
    return matArray;
}

// float** multiplyMatrices(float matA[][3], float matB[][3]){
float** multiplyMatrices(float** matA, float** matB){
  float** resultMat;
    resultMat = new float*[3];
    for (int i = 0;i<3;i++){
        resultMat[i] = new float[3];
    }
    for (int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            for(int k=0;k<3;k++){
                resultMat[i][j] += matA[i][k] * matB[k][j];
            }
        }
    }
    return resultMat;
}

// void applyTransform( pfs::Array2D *arrayX, pfs::Array2D *arrayY, pfs::Array2D *arrayZ, const fl
void applyTransform( pfs::Array2D *arrayX, pfs::Array2D *arrayY, pfs::Array2D *arrayZ, float** fin
{

  for(int i=0;i<3;i++){
          for(int j=0;j<3;j++){
```

```
            fprintf( stderr, "%i,%i: %4f \t", i, j, finalMatrix[i][j]);
          }
          fprintf( stderr, "\n");
  }

  int imgSize = arrayX->getRows()*arrayX->getCols();
  for( int index = 0; index < imgSize ; index++ ) {

    float &x = (*arrayX)(index);
    if( x < 0 ) x = 0;
    float &y = (*arrayY)(index);
    if( y < 0 ) y = 0;
    float &z = (*arrayZ)(index);
    if( z < 0 ) z = 0;

    x = finalMatrix[0][0]*x + finalMatrix[0][1]*x + finalMatrix[0][2]*x;
    y = finalMatrix[1][0]*y + finalMatrix[1][1]*y + finalMatrix[1][2]*y;
    z = finalMatrix[2][0]*z + finalMatrix[2][1]*z + finalMatrix[2][2]*z;

    // x = finalMatrix[0][0]*x + finalMatrix[0][1]*y + finalMatrix[0][2]*z;
    // y = finalMatrix[1][0]*x + finalMatrix[1][1]*y + finalMatrix[1][2]*z;
    // z = finalMatrix[2][0]*x + finalMatrix[2][1]*y + finalMatrix[2][2]*z;
  }

}

void applyTransformOnFrames( int argc, char *argv[] ){
    pfs::DOMIO pfsio;

  bool verbose = false;
    bool transpose  = false;
    bool opt_xyzrgb = false;
    bool opt_rgbxyz = false;

  // File name of Matrix file.
  char matrixFn[20];

    static struct option cmdLineOptions[] = {
    { "help",           no_argument,        NULL, 'h' },
    { "verbose",        no_argument,        NULL, 'v' },
    { "transpose",      no_argument,        NULL, 't' },
```

```
  { "xyzrgb",         required_argument,  NULL, 'x' },
  { "rgbxyz",         required_argument,  NULL, 'r' },
  { NULL, 0, NULL, 0 }
};

int optionIndex = 0;
while( 1 ) {
  int c = getopt_long (argc, argv, "r:x:htv", cmdLineOptions, &optionIndex);
  if( c == -1 ) break;
  switch( c ) {
  case 'h':{
    printHelp();
    throw QuietException();
  }
  case 'v':{
    fprintf( stderr, PROG_NAME " verbose is on\n" );
    verbose = true;
    break;
  }
  case 't':{
    transpose = true;
    break;
  }
  case 'x':{
    //char* matrixFn = optarg;
    fprintf( stderr, PROG_NAME " x option chosen\n" );
    strcpy(matrixFn, optarg);
    opt_xyzrgb = true;
    break;
  }
  case 'r':{
    strcpy(matrixFn, optarg);
    opt_rgbxyz = true;
    break;
  }
  case '?':
    throw QuietException();
    break;
  case ':':
    throw QuietException();
    break;
```

```
    }
  }

  float** matrixFile = 0;
  float** rec709Xuserval = 0;
  float** referenceMat = 0;
  matrixFile = readMatrixFile(matrixFn,transpose);

  if(opt_rgbxyz){
    referenceMat = initialiseArray(xyz2rgbD65Mat);
    rec709Xuserval = multiplyMatrices(matrixFile,referenceMat);
  } else if(opt_xyzrgb){
    referenceMat = initialiseArray(rgb2xyzD65Mat);
    rec709Xuserval = multiplyMatrices(referenceMat,matrixFile);
  }

  while( true ){
    pfs::Frame *frame = pfsio.readFrame( stdin );
    if( frame == NULL ) break; // No more frames

    pfs::Channel *X, *Y, *Z;
    frame->getXYZChannels( X, Y, Z );

    // TODO : Processing here
    if( X != NULL ) {           // Color, XYZ
      applyTransform( X, Y, Z, rec709Xuserval);
    } else
    throw pfs::Exception( "Missing X, Y, Z channels in the PFS stream" );

    pfsio.writeFrame( frame, stdout );
    pfsio.freeFrame( frame );
  }
}

int main( int argc, char* argv[] )
{
  try {
    applyTransformOnFrames( argc, argv );
  }
  catch( pfs::Exception ex ) {
    std::cerr << PROG_NAME << " error: " << ex.getMessage() << std::endl;
```

```
    return EXIT_FAILURE;
  }
  catch( QuietException  ex ) {
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Color Calibrated High Dynamic Range Timelapse Using Remote Capture

Aim

The aim of this thesis is to produce a system capable of making timelapse videos with very accurate colors.

Timelapse photography is used to take photos of a scene at a set interval (often 2-10 secs.) and then played at a much faster frame-rate than the capture interval to visualise the process at a much faster rate than its occurrence.

The camera is calibrated before taking photos and the photos are captured using HDR technique (see right).

The timelapse is started remotely from an Android device.

High Dynamic Range photography is used to capture the most accurate representation of the scene. as seen by the human eye. By merging photos taken at different exposures (by varying shutterspeed, ISO or aperture), it ensures there are no under-exposed or over-exposed pixels.

Why is calibration required?

Calibration is required to produce a device independant reproduction of a scene. Any device that captures or displays images should be calibrated so that the colors it shows are representative of the originally intended color. The image in Fig shows the same image on two different screens. Some of the colors are noticeably different (esp. on Row 3).

For calibration, three cameras were tested. As seen in Fig 3, before calibration, captured pics. had differences in colors. However, after calibration images produced by all 3 cameras look perceptually similar.

Results of color calibration were published in a paper[1].

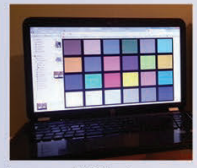


Image on TFT laptop screen.

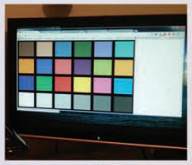


Image on LCD screen.

Fig 3: Scene before and after Calibration using 3 cameras.

1.Colorimetric calibration of high dynamic range images with a ColorChecker chart, Delvin Varghese et. al, proceedings of HDRi 2014: Second International Conference and SME Workshop on HDR imaging, Sarajevo, Bosnia.

Remote Capture Setup

The setup involves a Canon 1000d DSLR camera connected to a raspberryPi, a portable single-board computer which is running a server. The user initiates the timelapse from an android app (programmed in Eclipse IDE.) The app sends the capture settings to the server program which then captures the HDR exposures onto the camera's storage by calling a program called `gphoto2`

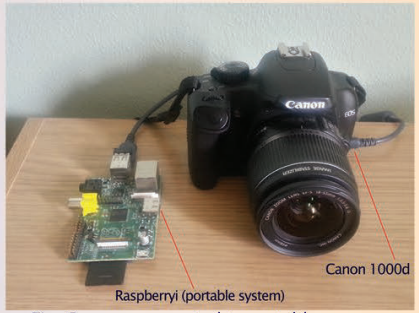


Fig: Camera connected to portable server.

Fig: Android App.

User can press Preview button to capture and retrieve an image from the camera.

Timelapse Generation

Once the server program has finished capturing the required exposures, the user then connects the camera to a PC and executes a shell script which will:

1) Merge exposures into their respective HDR file by calling a command line program `pfstools`.

2) Merge the HDR files into a video file using the command line program `mencoder`. The video is made at 20 frames per second i.e. 20 HDR files are used to generate 1 second of video.

Fig: Timelapse Video

Evaluation

Using HDR images in a timelapse works best when the changes being monitored are slow e.g. movement of clouds, tracking the sun at sunset/sunrise etc.

However, it has drawbacks because each shot requires a number of shots be taken. For scenes with moving objects e.g. humans, this will result in blurred images or Ghosting, which results in bad frames in the timelapse video.

Another problem that was encountered was the time it takes for the software to execute a camera-capture command: 4-5 seconds. For a 3-exposure HDR timelapse, each frame will take 12-15 seconds to capture. Unfortunately, there is no fix for this at the moment.

Delvin Varghese Supervisor: Rafal Mantiuk

Figure 7.5: Dissertation Poster