TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

HEIKKI PARVIAINEN

ANDROID CAMERA TUNING APPLICATION

Master of Science thesis

Examiner: Prof. Timo Hämäläinen
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on April 6th 2016

ABSTRACT

HEIKKI PARVIAINEN: Android camera tuning application
Tampere University of Technology
Master of Science thesis, 43 pages
May 2016
Master's Degree Programme in Electrical Engineering
Major: Embedded Systems
Examiner: Prof. Timo Hämäläinen
Keywords: field camera tuning android application

The Image Quality team at Intel had a problem with the tuning tools being tied to not enough mobile tool implemented with Matlab. The final tuning of the camera requires lot of field testing and carrying a device capable of running the Matlab tool along with the device under test is not practical.

To solve this issue an Android application capable of changing the camera tuning parameters was created. The application does this by changing a tuning file located in the Android device's file system. The application was verified to provide the same tuning file as the Matlab tool.

Android platform was chosen as Android is the most requested platform for Intel's customers. The Image Quality team for which this application is made was very happy with the results. The application is still in development for new features.

TIIVISTELMÄ

HEIKKI PARVIAINEN: Android-kameran virityssovellus
Tampereen teknillinen yliopisto
Diplomityö, 43 sivua
Toukokuu 2016
Sähkötekniikan koulutusohjelma
Pääaine: Sulautetut järjestelmät
Tarkastajat: Prof. Timo Hämäläinen
Avainsanat: kenttä kamera viritys android applikaatio

Intelin Image Quality tiimillä oli hankaluuksia liikuteltavuuden kanssa, koska heidän kameran viritys työkalu on kirjoitettu Matlabilla. Virityksen loppuvaiheissa tarvitaan paljon testausta kentällä ja Matlabiin kykenevän laitteen kantaminen testattavan laitteen kanssa on hyvin hankalaa.

Tätä ongelmaa ratkaisemaan tuotimme Android ohjelman, joka pystyy muokkaamaan kameran virityksessä käytettyjä parametreja. Ohjelma muokkaa viritystiedostoa joka sijaitsee Android laitteen tiedostojärjestelmässä. Työkalu varmennettiin tuottamaan sama viritystiedosto Matlab työkalun kanssa.

Android valittiin alustaksi, koska se on eniten kysytty alusta Intelin asiakkaiden keskuudessa. Image Quality tiimi, jolle ohjelma tuotettiin, oli hyvin tyytyväinen tulokseen. Ohjelmaa kehitetään yhä ja siihen tuotetaan uusia ominaisuuksia.

PREFACE

I would like to thank Laura Moisio for her invaluable help with the proofreading. Also my manager Antti Stenhäll and all the other people at Intel Tampere who did their best to encourage me do and finish this thesis. In the end when I just got to doing this, it was not so hard. Thank you also to my advising professor Timo D. Hämäläinen.

Bali, 20.7.2016

Heikki Parviainen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3A+	Composite term for AF, AE, AWB and other control algorithms
ADB	Android Debug Bridge
AE	Auto Exposure
AF	Auto Focus
AWB	Auto White-balance
CAF	Continuous Auto Focus
CCT	Correlated Color Temperature
CFA	Color Filter Array
CI	Continuous Integration
CIE	International Commission on Illumination
CPF	Camera Parameter Framework
FOW	Field of View
HVS	Human Visual System
IQ	Image Quality
MACC	Multi Axis Color Control
RGB	Red Green Blue
UI	User Interface
WOI	Window of Interest
WP	White Point

1. INTRODUCTION

Currently at Intel the all of the camera tuning is done with the Image Quality (IQ) Tool which is written with Matlab. This tool needs a PC to run so the tuning engineer is bound to his computer during the tuning process. In the final tuning process there is a need for a tool to do tuning on the field and carrying a device capable of running Matlab along with the device under test is not practical.

The purpose of this project is to produce an Android application capable of changing the tuning file. The Android application will not try to provide same functionality as the IQ Tool, but the target is to find the most important parameters the tuning engineers would need on the field. As the tuning file structure keeps evolving while also keeping backwards compatibility the Android application needs to be easily extended to accommodate new content added to the tuning file.

The IQ Tool tool also requires a lot of training before it can be effectively used. The application was created to be extendable, but a version of the application stripped of parameter targeted for Intel's customers can also be released. Intel will do the initial tuning, but as the customer usually wants to be part of the tuning process the Android application will provide them with a simplified way to do this.

2. CAMERA BASICS

The goal of a camera system is to be able to capture a scene and later to reproduce the captured scene. To achieve this goal much processing is done between light incoming to the sensor and light coming out of the devices screen.

2.1 Sensor

The camera sensor is the first component in the camera system. It consists of a lens, a color filter and the image sensor. The lens focuses light through a color filter into the image sensor which is a matrix of light sensitive elements called photosites as presented in Figure 2.1.

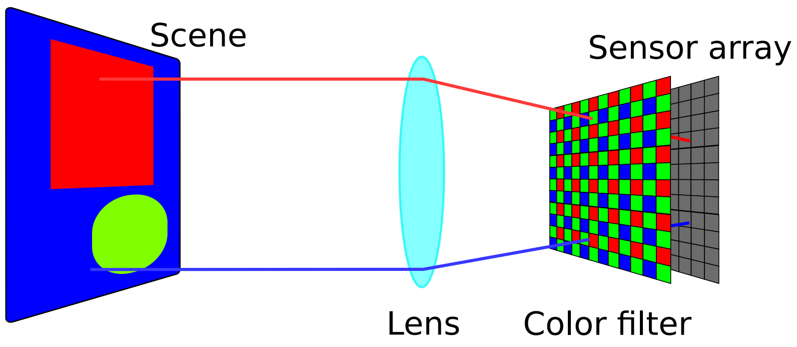


Figure 2.1 *Camera sensor components.*

The lens is actually a collection of glass or plastic elements which together shape the light to fit the sensor and provide the final focus point. To enable different focusing of the image the final lens element is connected to a motor which allows the element to move in respect to the sensor.

Each of the photosite in the image sensor are identical so to capture color images a filter is applied over each pixel. This element is called color filter array (CFA). Figure 2.2 presents a few different possible CFAs where the Bayer filter and its derivatives are the most common. The Bayer filter uses twice as many green pixels as blue or red because the human eye is most sensitive to green.

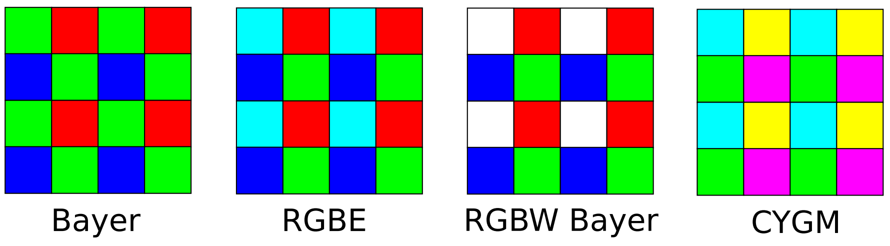


Figure 2.2 *Different color filter arrays*

Each of the sensor elements, consisting of a color filter over the sensor pixel, contain luminance data that has been altered by the filter. A full-color image can be constructed from the sensor elements' raw data by a demosaicing algorithm [1]. Figure 2.3 presents three different states of image capture. First section is the original image, the second is the output of the sensor array, the third section has the sensor array output color code with the Bayer filter colors and the fourth is the image reconstructed by simple interpolation.

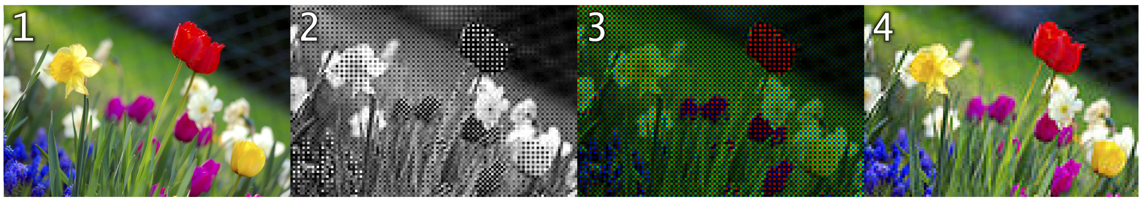


Figure 2.3 *1: Original image 2-3: Bayer raw image 4: Image reconstructed by interpolating*

Reconstruction is needed as each pixel contains exact measurement of only one of the three main colors. The remaining two color values are interpolated from the neighboring pixels. This approach works well as long as the image does not contain sharp edges where it can cause artifacts such as color bleed over the edge. Usually more sophisticated algorithms are used in the image pipeline.

The sensor array can be considered as an array of buckets (photosites) that collect photons and transform them into voltage value. Before starting to take the picture each bucket is first emptied and after time period called the exposure time the voltage value is read.

The difference between the largest and the smallest signal that is usable for a system is called the system's dynamic range. For a camera sensor the dynamic range is set by the black and white level of the photosite. The black level is the value a photosite will give when no light is sent into the sensor, a picture of complete darkness. The black level is limited by noise caused by electrical interference such as leak currents and imperfections of the sensor [2]. White level is the value where the photosite is saturated and the value will not go any higher even if more light enters the sensor. The black and white levels are visualized in Figure 2.4. Typically the white level and the dynamic range increase when increasing the sensor size as each the area of each photosite also increases.

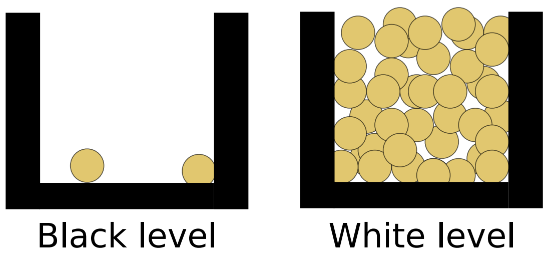


Figure 2.4 *Dynamic range*

The voltage values from the photosites are converted into binary values by a analog to digital converter. Typical conversions are usually 10-bit up to 14-bit which produces numbers from zero to 16 364. Increasing the bit depth of the analog to digital conversion does not increase the dynamic range but the sensor can produce finer distinctions withing the dynamic range. The dynamic range can be considered as the size of the stair as the bit depth as the size of one step. Figure 2.5 presents this analogy.

Sensors that are targeted for the mobile space usually do not have a physical shutter. A common way of resetting the sensor array when capturing full image is using a method called the rolling shutter. This method scans across the array either vertically or horizontally reading and resetting each row or column as it proceeds.

The rolling shutter can cause distortions in the image as different parts of it are recorded at different times. These effects are visible only when the scene changes rapidly. Some common distortions are for example skew where an object bend diagonally if the camera or the object moves horizontally or partial exposure where lighting conditions change during the capture. Figure 2.6 depicts these distortions.

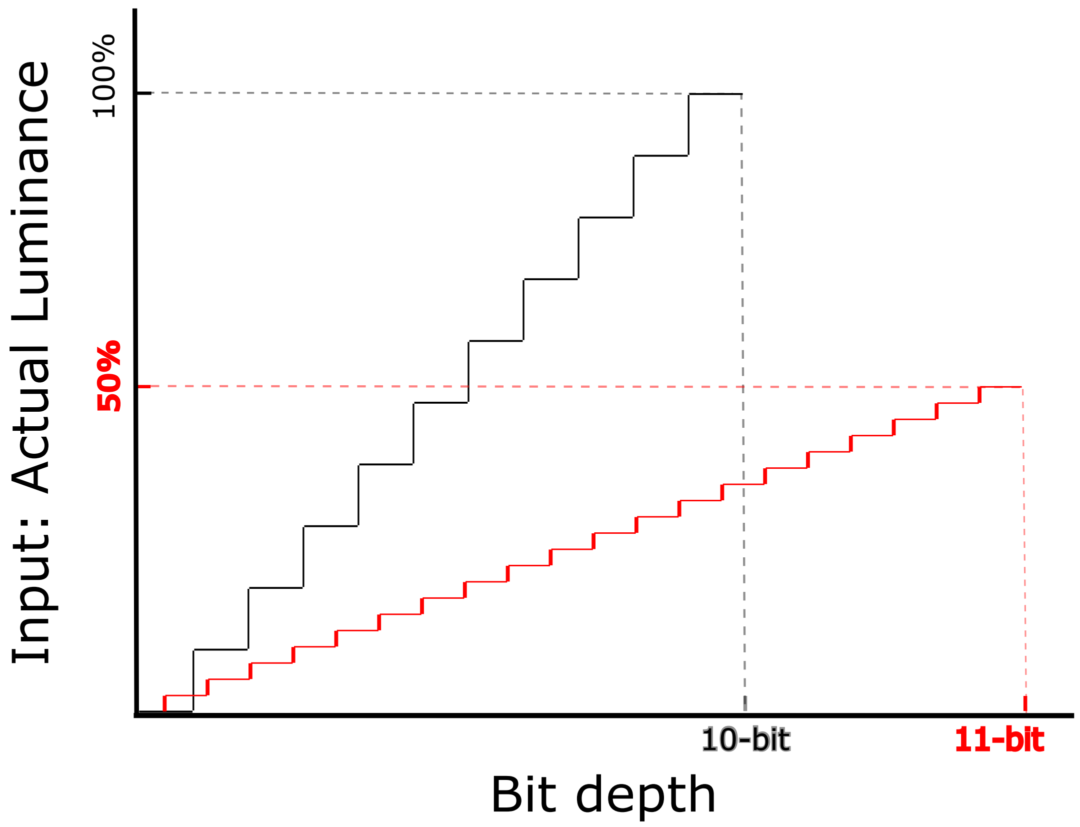


Figure 2.5 *Visualization of bit depth and the dynamic range*

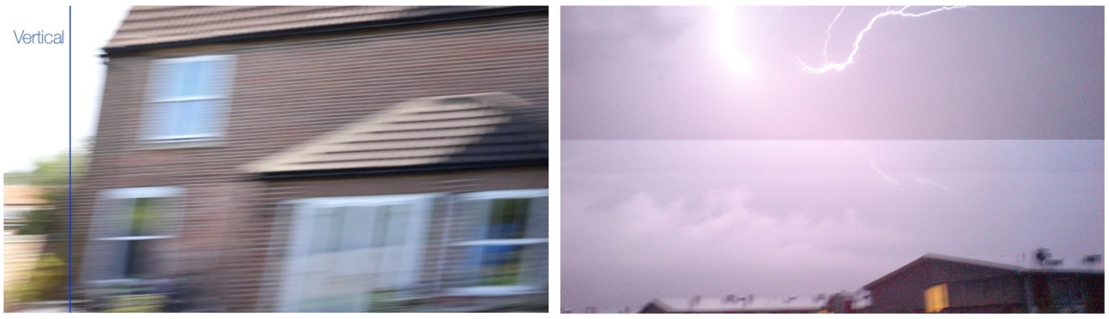


Figure 2.6 *Distortions caused by the rolling shutter*

2.2 Image pipeline

The image pipeline gets the raw image as input from the image sensor and its purpose is to reconstruct the original scene from the input. Figure 2.7 presents a flow chart of the image processing pipeline.

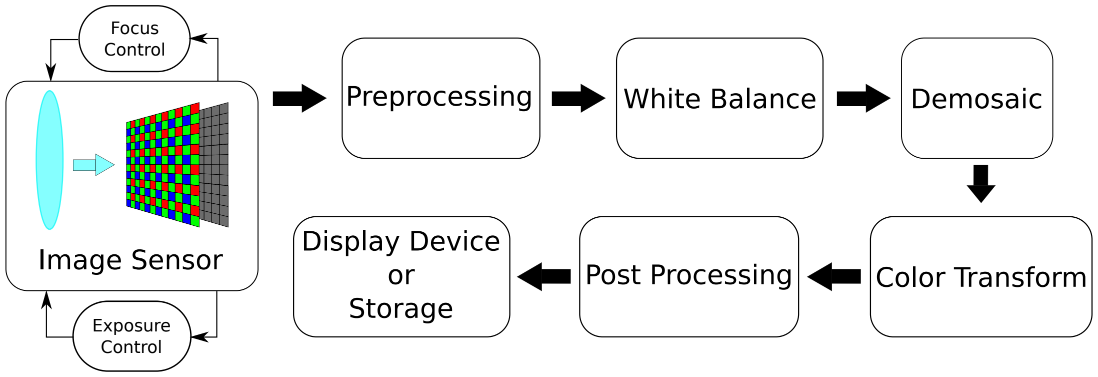


Figure *2.7 Image pipeline*

The focus and exposure controls can be both derived from the actual luminance derived from the red, green and blue (RGB) image or simply from the green channel data of the sensor which is a close estimate of the former. The exposure control analyses the brightness of taken images and loops back to the sensor with adjustments to the exposure time, gain controller and aperture size although mobile sensors usually have fixed aperture size.

Focus control methods can be divided into passive and active methods. Passive approaches for focus control analyzes the spatial frequency content of the image data [3]. Finding high frequency content in the area of interest means the image is in focus as high frequency means sharp edges. The process is then iterated while changing the focus between iteration until desired focus is found. Passive focus control does not require additional hardware but because of the iterative nature is slower than the active approach. Active focus control methods typically use a beam of infrared light sent from a source near the image sensor to get a a distance estimate of the object of interest [4].

Before the raw data from the sensor is further processed some preprocessing is required. On this step issues caused by the sensor are compensated. The sensor might contain defective photosites which commonly have stuck to zero or the maximum value. These defective pixels are corrected by estimating their value from correct neighboring pixels. Other possible preprocessing steps are linearizion if the sensor has nonlinearities or subtracting the black level image from the captured data.

When white piece of paper is captured under different lighting conditions such as fluorescent light or natural daylight the color perceived by the sensor differs for different illuminations. White balance is the process of adjusting the image so that white colors appear truly white and that the colors are accurate under all light sources. Figure 2.8 presents same image with different white balance settings. The image pipeline must determine the color temperature of the light source and then add or subtract color to correct the distortion caused by the light source.

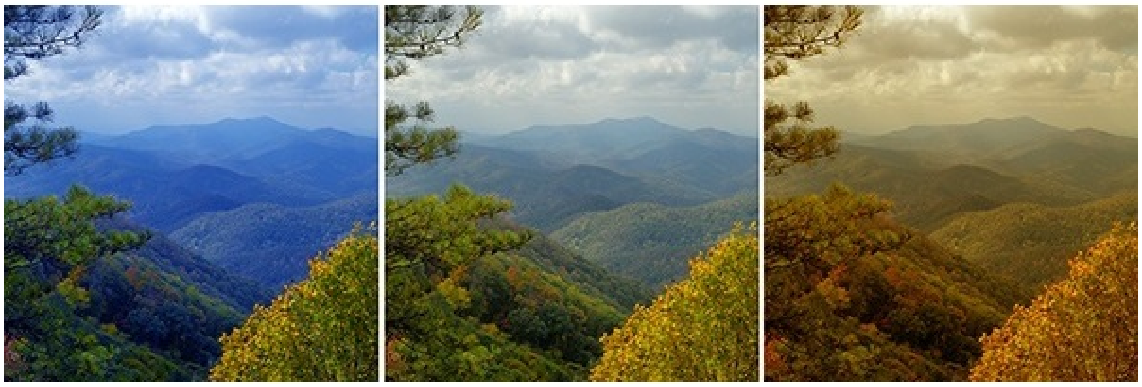


Figure 2.8 *Image with different white balance settings [5].*

To calculate the RGB color information in the image a demosaicing step is required as each pixel contains only one color value due to the CFA. It is the most computationally exhaustive part of the image pipeline [6]. All of the the methods for demosaicing use information from the neighboring pixels to estimate the pixel colors that were not measured and at the same time try to avoid introducing artifacts into the image.

The human eye contains three types of color sensitive cells called cone cells which are active under medium and high brightness conditions. In low light conditions color vision diminishes and the monochromatic rod cells activate. The spectral sensitivity of the cone cells peaks at wavelengths corresponding to short (S, 420-440 nm), middle (M, 530-580) and long (L, 560-580) wavelengths as presented in Figure 2.9 [7].

The camera sensor does not have the same spectral sensitivity as the human eye so to help with the further computing the image is transferred to a different color space. An example of such color space is the Internal Commission on Illumination (CIE) 1931 XYZ color space that defines quantitative links between human color vision and wavelengths in the visible electromagnetic spectrum.

Before the end of the pipeline the image goes through post processing where the

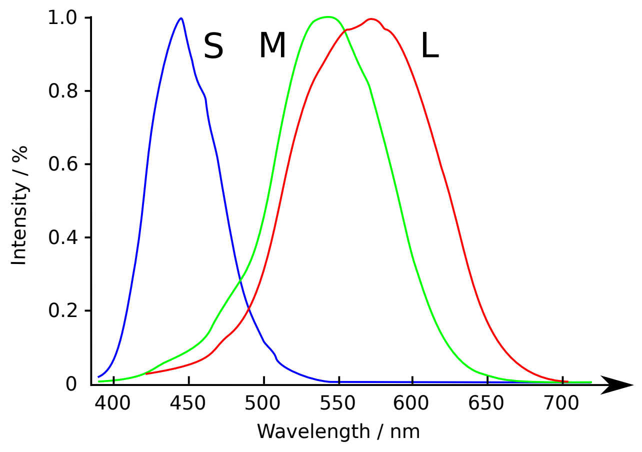


Figure 2.9 *Normalized spectral sensitivity of human cone cells.*

possible artifacts introduced by the previous steps are corrected. The demosaicing step for example may introduce a zipper type artifact along the edges with strong intensity [8]. A few of the common post processing steps are color artifact removal, edge enhancement and coring[1]

Psychophysical experiments show that an image with more accentuated edges is subjectively more pleasing [9]. Edge enhancement aims to locate and enhance edges in an image. It does so by using plethora of mathematical tools such as gradients and adding the Laplacian to the image. Figure 2.10 presents an image before and after edge enhancement.

At the end of the pipeline the image is compressed. Usually the compression is done by transforming the image into sRGB color space. The sRGB is a widely spread color space standard used by virtually all output devices. In the transformation the original 10-14 bit data values are interpreted into 8bit values causing it to be a lossy transformation. Figure 2.11 presents the sRGB color space within the CIE 1931 XYZ color space. As seen in the figure the sRGB color space is not able to produce nearly as many colors as the CIE 1931 XYZ color space.

[1]In goring image information that does not contribute to the image detail and behaves like noise is removed.

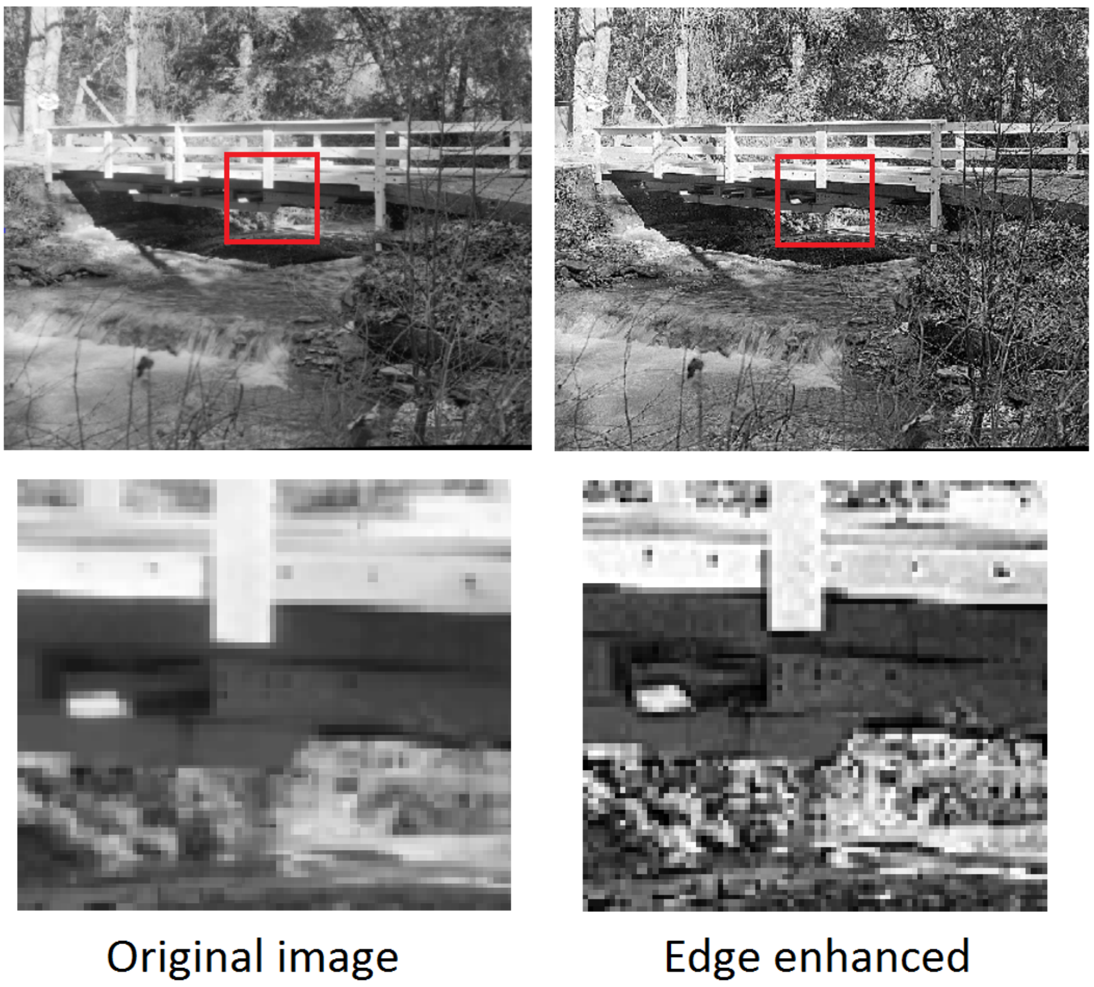


Figure 2.10 *Before and after edge enhancement[10].*

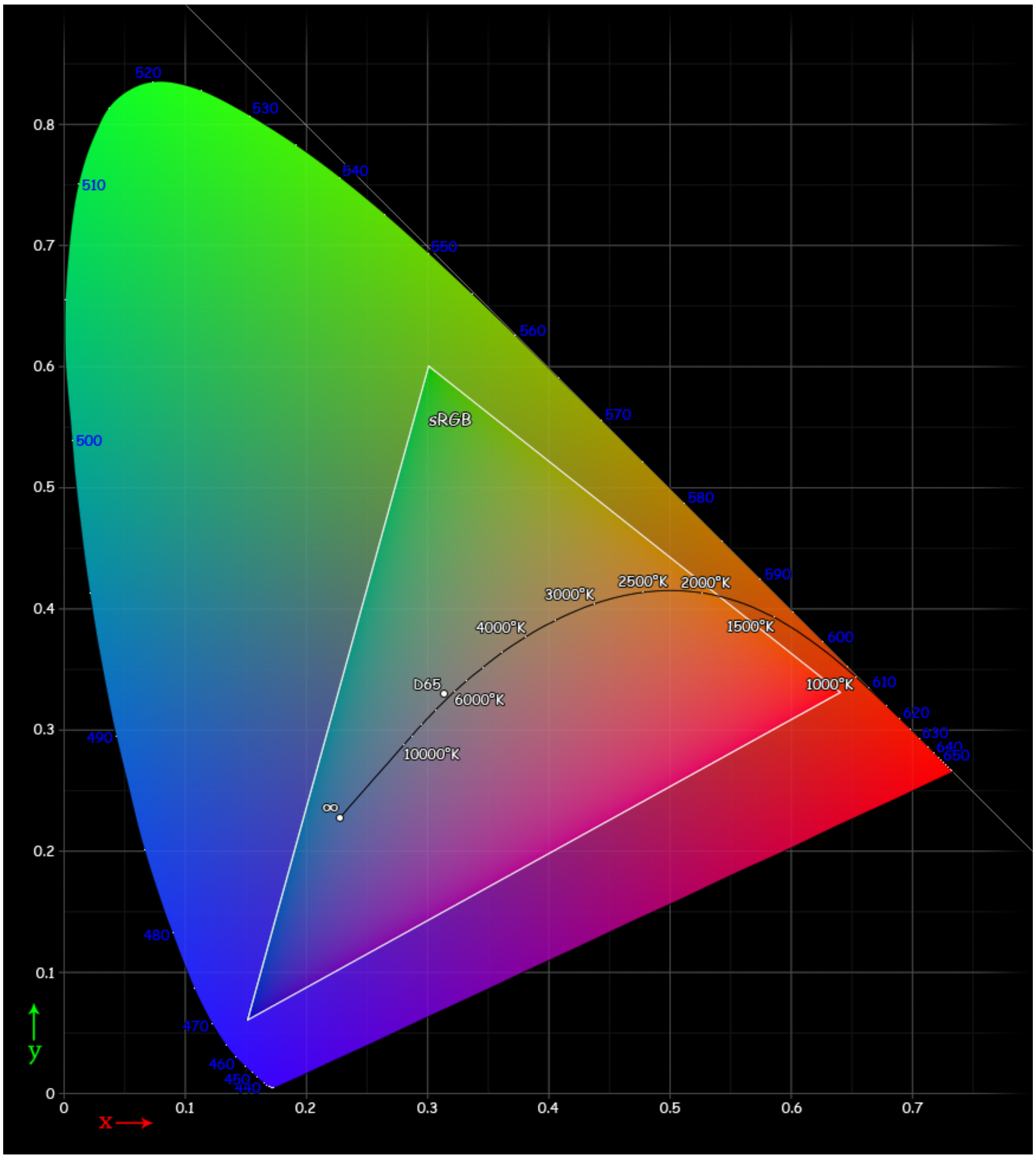


Figure 2.11 *The sRGB color space inside the CIE XYZ color space [11]*

3. CAMERA TUNING

The camera system perceives a scene in different way than the Human Visual System (HVS) does. The camera sensor imposes weaknesses that need to be understood to produce recognizable and visually pleasing images. The HVS also includes nonlinearities and limitations that prevent us in perceiving a scene perfectly. The human factor also means that a good tuning is ultimately a matter of personal taste.

Camera tuning can be roughly separated into two parts. First the sensor module characterization and the final tuning.

3.1 Characterization

The camera sensor's capabilities vary depending on numerous factors, such as quality of the silicon and the color filter or the optics. There are variations even between sensors of the same model caused by the manufacturing process. In characterization the sensors properties are studied so that it is understood how the sensor perceives visible light (i.e. white balance and colors), brightness (i.e. sensitivity) and to know weather the sensor has systematic issues that need to be fixed (i.e. black level and lens shading). The information is gathered by taking many raw pictures in controlled settings so that it is possible to analyze and compare what the sensor captured with what was the actual scene.

3.1.1 Sensor sensitivity

The sensor sensitivity value is used to estimate the illumination level on a scene. To asses sensors sensitivity images of a surface for which an accurate illumination is known are captured. With the raw images that are not overexposed, the known surface brightness and the exposure time the sensor sensitivity can be calculated. If the sensitivity is incorrectly defined, systems like Auto Exposure (AE) and Auto White Balance (AWB) may not function correctly.

3.1.2 Black level and Saturation level

The black level is measured by capturing raw images with varying gain levels in total darkness and the saturation level is measured by capturing in bright light while applying very long exposures. Analyzing these images provides the minimum and maximum pixel values provided by the sensor. If the black level is set too high resulting images have poor contrast in black tones. If it is set too low images may contain reddish tone. Wrongly set saturation level causes the images to have poor contrast or lack of bright tones.

3.1.3 Sensor linearity

In a perfect sensor the separate color channel pixel values act linearly through the dynamic range, but in practice the channels start acting non-linearly when the exposures are 80%-95% of the dynamic range. The AE is then adjusted so that the non-linear part of the sensors dynamic range is not utilized.

3.1.4 Lens shading

Lens shading or vignetting causes the edges of the image appear darker compared to the center. The Figure 3.1 demonstrates the causes of lens shading.

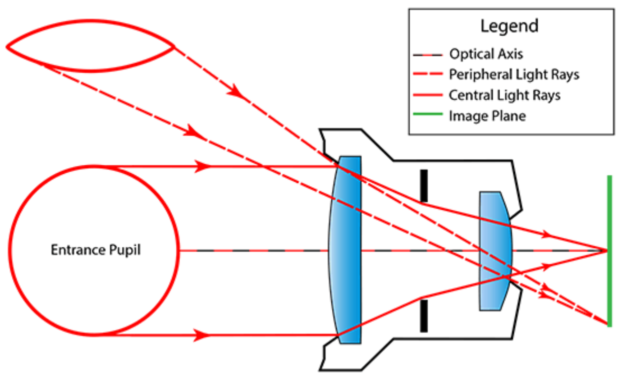


Figure 3.1 *Lens shading [12].*

The edges of the sensor module can block the part of the incoming light that comes in with an angle. Also when light travels through the lens the rays at the edges travel longer distance than rays coming in at a right angle. As all the sensor's pixels point the same way in a flat plane, the pixels on the edges receive less light as the light is coming in on an angle.

Lens shading is measured by taking completely flat images (images that don not have any contrast whatsoever) for different light sources by integrating a diffuser glass in front of the sensor. The flat images demonstrate how much each color channel attenuates when moving toward the edge.

3.1.5 Auto white balance and Chromaticity response

To tune AWB and chromaticity response raw images of a standardized color chart are taken for as many light sources as possible. Figure 3.2 presents a common color chart used in industry called the Macbeth chart. The tiles of a physical Macbeth chart are designed to maintain color consistency by guaranteeing to reflect light in the same way [13].

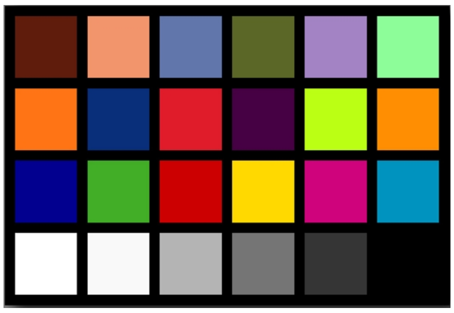


Figure 3.2 *Macbeth Color Checker Chart.*

For AWB a sensor module that best represents the average of all the modules needs to be selected. The calibration images of the color chart and a corresponding flatfield image with a diffuser are taken. With the calibration images the grey achromatict tiles (bottom row on Figure 3.2) are analyzed in chrominance space. The R, G and B channels are presented in R/G and B/G where the area of achromatic surface is approximated from the calibration images.

Figure 3.3 represents the achromatic surface approximation in the [R/G, B/G] space. The automatic white balance algorithm splits the taken image into a grid and calculates where the area inside each grid is located in the space. The light source can be identified when some of the grid values fall inside the achromatic area.

The chromacity response calibration is similar to the AWB calibration. In it the

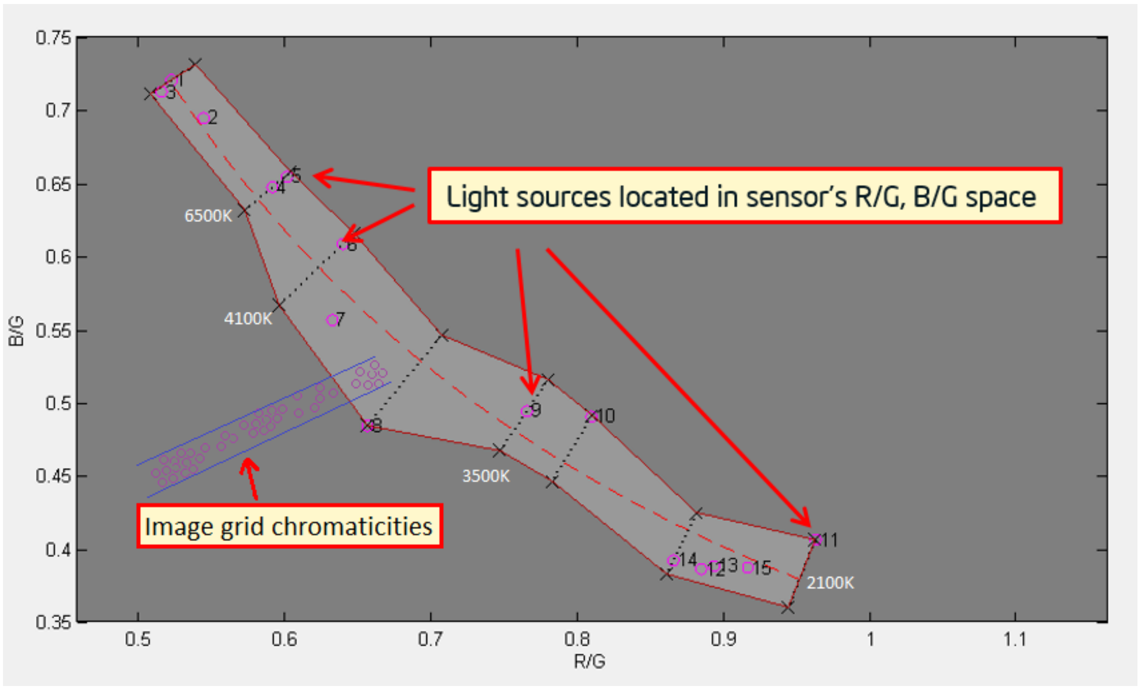


Figure 3.3 *The response of a typical sensor for different light sources in sensor chromaticity space.*

color tiles are used in addition to the gray achromatic tiles. The analysis produces a color conversion matrix that is used in the demosaicing step.

3.2 Final tuning

After the characterization the camera is able to reproduce accurate colors, identify the color temperature of the scene, find correct white balance and estimate the brightness level. Further actions include:

- Improving the AWB to correctly operate under various environments.
- Defining exposure and gain guidelines and setting exposure parameters.
- Setting the autofocus specific parameters.
- Carrying out ISP tuning for filtering, sharpening, etc.

As the final tuning is the target for the Android application we will delve deeper into this subject in Chapter 5: Tuning Algorithms.

4. SOFTWARE IMPLEMENTATION

The purpose of this software is to help the tuning engineers by providing more mobility in the final tuning.

The application is implemented as a Android's system alert window so that the application is presented always on top. This way the tuning engineer can view the camera application and see the changes in real time.

The application provides the ability to change predefined fields of the configuration file or quickly switch between configuration files that are uploaded to the device through the Android Debug Bridge (ADB).

Figure 4.1 presents the screen where the user can choose which parameter they want to configure. A touch event will open the controls for that particular parameter. The user can come back to this screen by pressing the wrench icon. When the load icon is pressed the screen in Figure 4.2 opens. This screen presents all files under a predefined folder inside the devices file system and pressing any will load it as the current one.

The update icon will reset the camera pipeline which will also update the edited configuration file. The cross icon will close the application. The application need its own shut down mechanism as the system alert window means that it does not follow the usual Android application life cycle.

4.1 User Interface

When the user chooses a parameter to tune from the list in Figure 4.1 a tuning User Interface (UI) is shown. The tuning UI can be a list of sliders as presented in Figure 4.3. This is a a generic UI for most of the parameters where each slider presents one integer value. In some cases when different values depend on each other so when we change one value the other one should change with it. This does not require changes

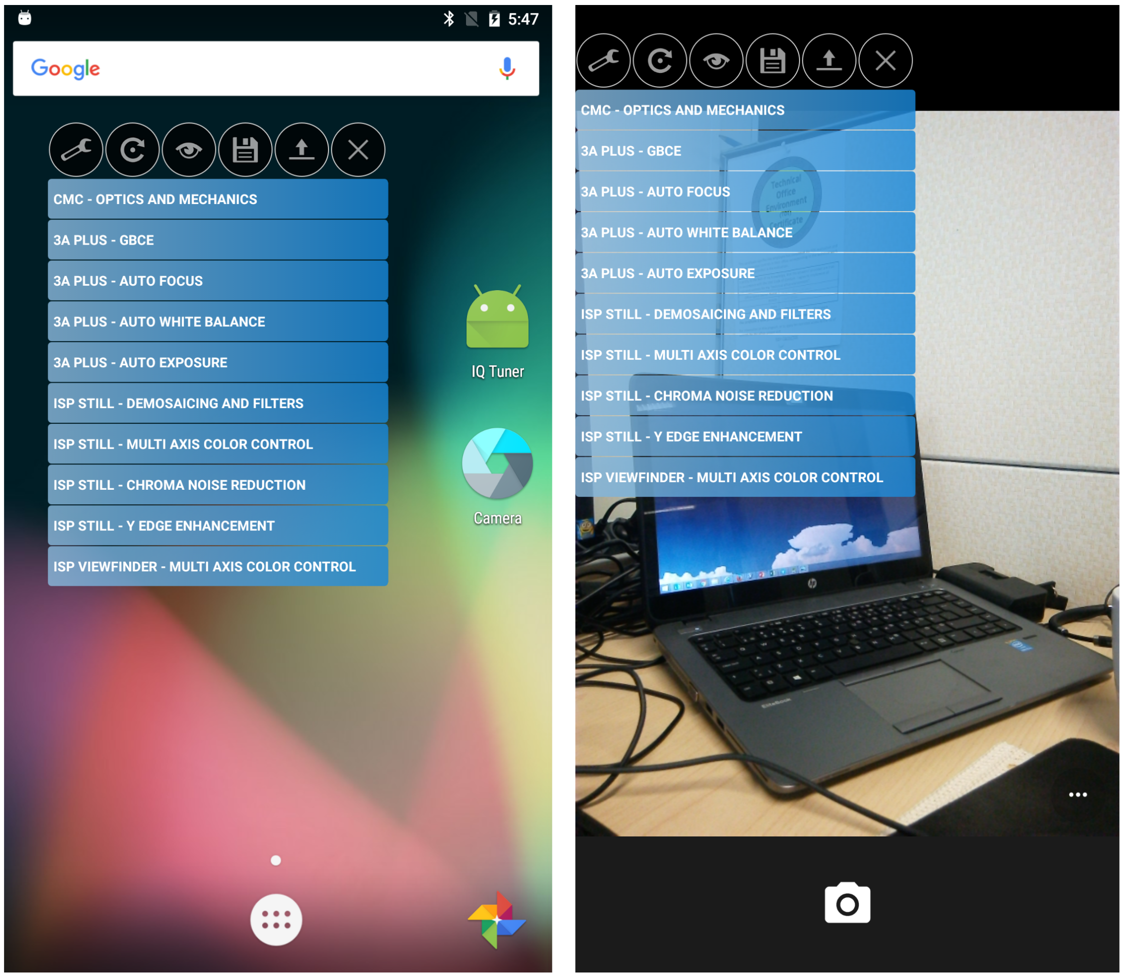
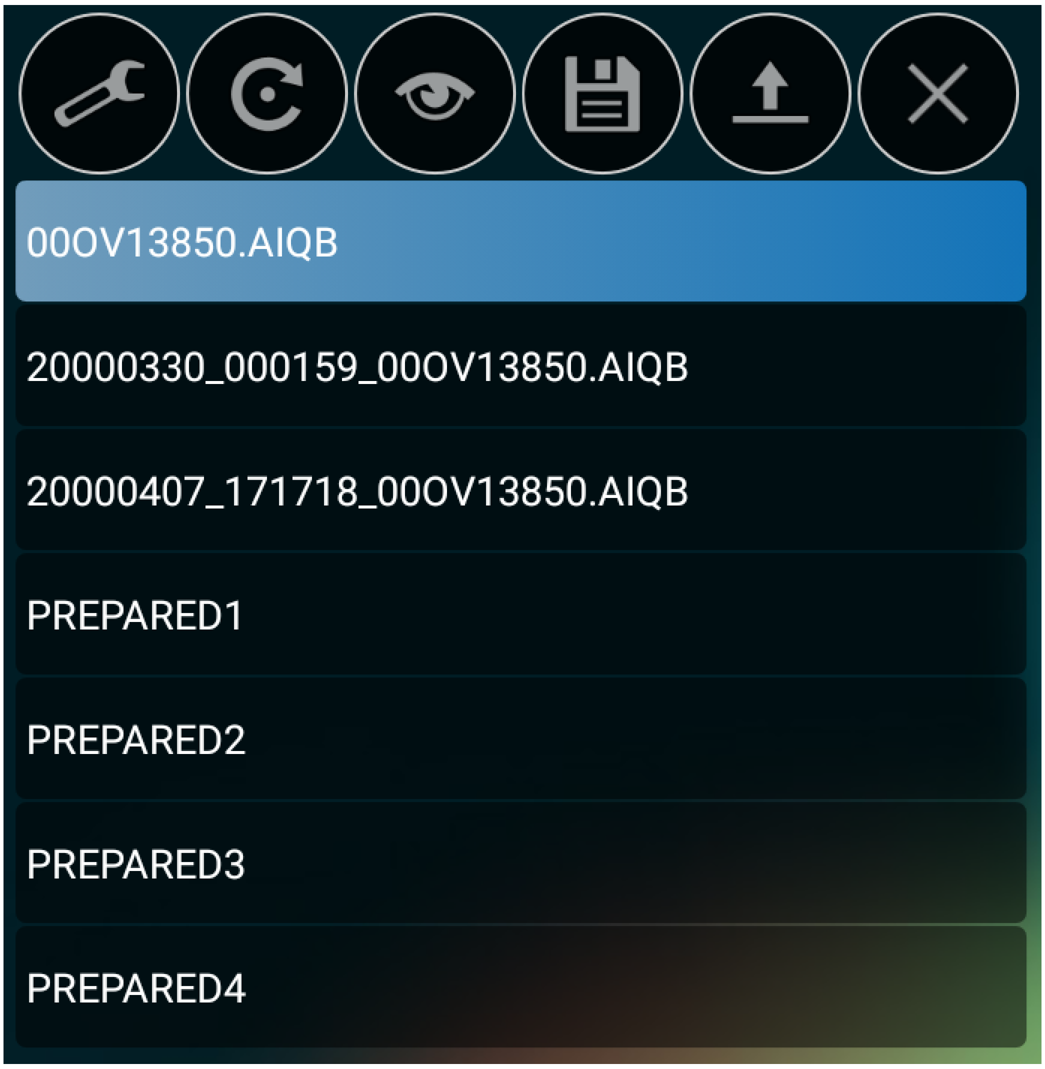

5:47
Google
CMC - OPTICS AND MECHANICS
3A PLUS - GBCE
3A PLUS - AUTO FOCUS
3A PLUS - AUTO WHITE BALANCE
3A PLUS - AUTO EXPOSURE
ISP STILL - DEMOSAICING AND FILTERS
ISP STILL - MULTI AXIS COLOR CONTROL
ISP STILL - CHROMA NOISE REDUCTION
ISP STILL - Y EDGE ENHANCEMENT
ISP VIEWFINDER - MULTI AXIS COLOR CONTROL
IQ Tuner
Camera

Figure 4.1 *Start screen of the application.*

to the generic UI, but the underlaying implementation need to handle this.

In some cases a more sophisticated UI is required, such as the Multi Axis Color Control (MACC). The MACC data structure contains 16 tables each having four integers to a total of 64 values. We created a custom UI to change these values using the Android Canvas class to draw and capture touch events for control. The MACC UI is presented in Figure 4.4.



000V13850.AIQB

20000330_000159_000V13850.AIQB

20000407_171718_000V13850.AIQB

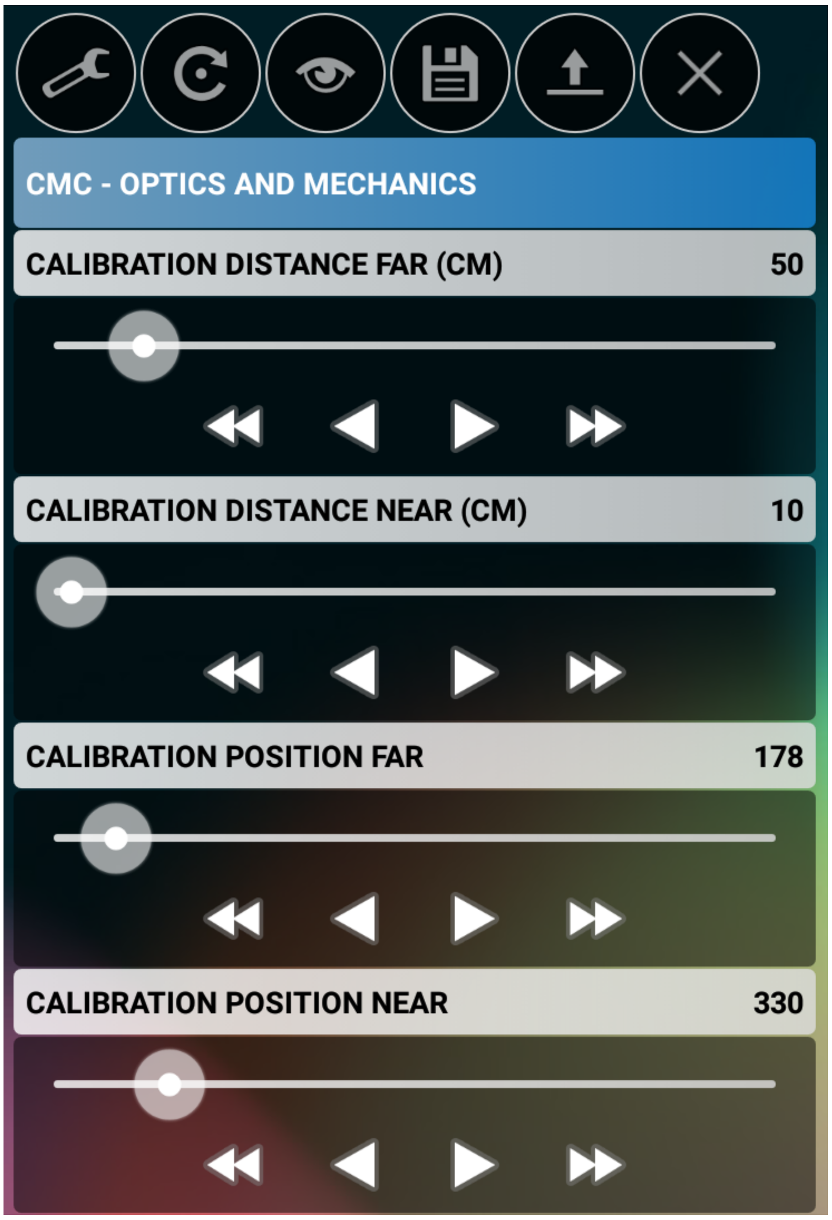
PREPARED1

PREPARED2

PREPARED3

PREPARED4

Figure 4.2 *File selection screen.*


CMC - OPTICS AND MECHANICS

CALIBRATION DISTANCE FAR (CM) 50

CALIBRATION DISTANCE NEAR (CM) 10

CALIBRATION POSITION FAR 178

CALIBRATION POSITION NEAR 330

***Figure** 4.3 Generic slider user interface.*

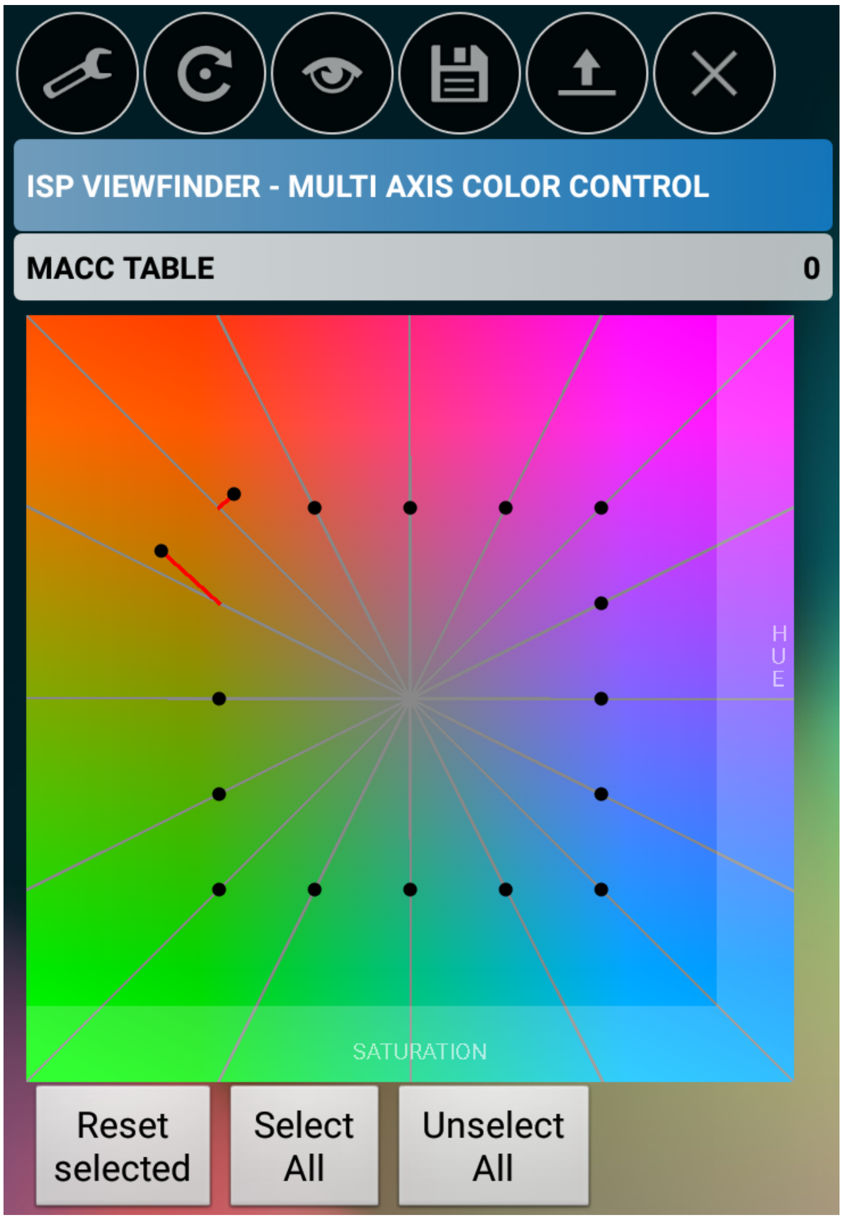


Figure *4.4 Custom interface.*

4.2 Implementation

In this project Google's official integrated development environment Android Studio was used for developing as it contains all tools from building to debugging. As the application is implemented as a system alert window the main activity class only acts as a creator for this service. The main activity also checks and asks the user for permission to draw overlays. Figure 4.5 presents the aplication's high level class overview.

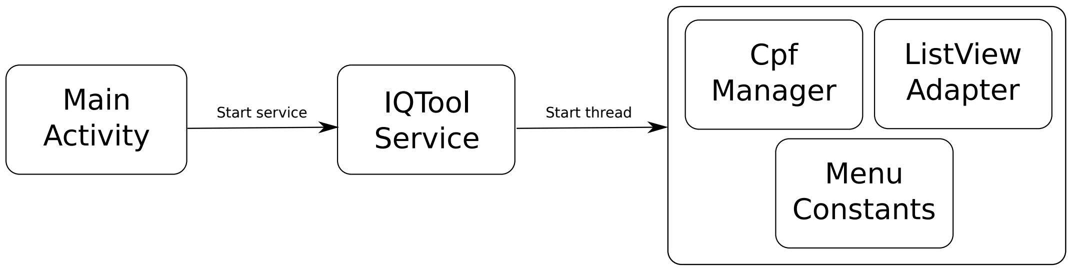


Figure 4.5 *Class overview.*

The system alert window service is implemented in class IQToolService. It is basically the same as the main activity in a normal Android application as it contains the root view and all the other objects are initialized here. It starts a thread running objects that parse the configuration file, handle UI events and has a menu constants class.

4.2.1 CpfManager

The CpfManager (Camera Parameter Framework Manager) class handles the reading and writing to the configuration file. It reads the file collecting offsets to start of all the headers within the file. If the ID inside a header is contained in the MenuConstants class the CpfManager calls a constructor for a class that does further parsing.

As presented in Figure 4.6 parser for each ID needs to be implemented separately as only the header contains data that is in the same format for all IDs. All of the parsers inherit a common class for functions that are shared for all IDs. The type of data contained within each ID is specified in the original IQ Tool and in excel sheets.

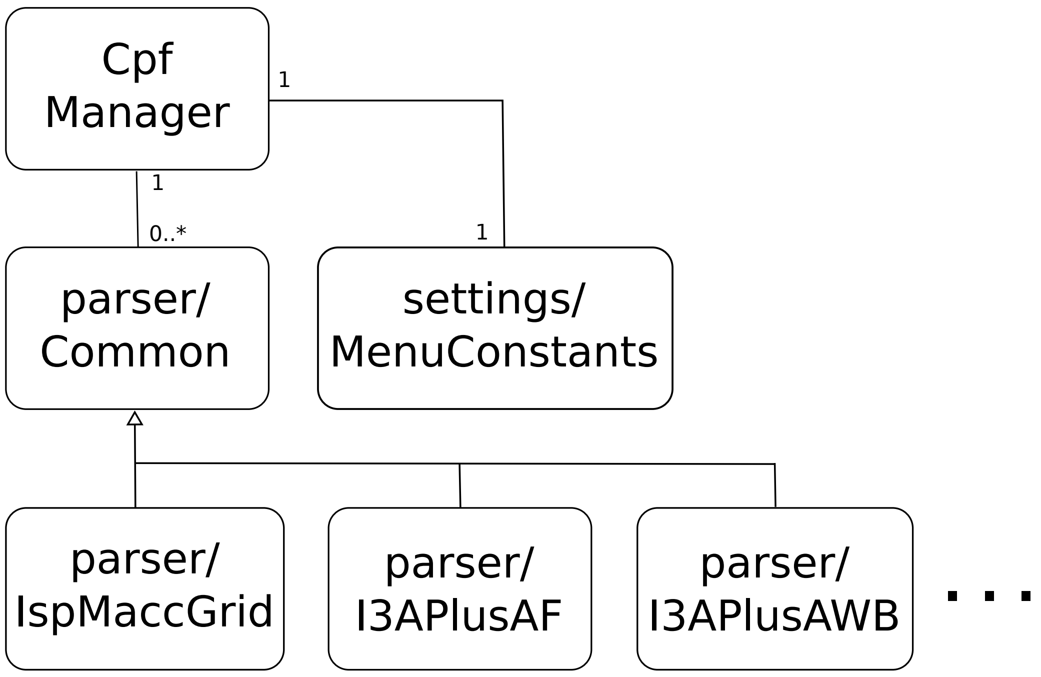


Figure 4.6 *CpfManager class diagram.*

An important functionality of the parser is keeping count of the checksum in the configuration file. The checksum is calculated by reading through the binary file signed integer (four bytes) at a time and adding them together ignoring any overflows. This method was also tried but it was too slow or took too much memory. If the checksum is calculated in a loop with file I/O the read operations take too long and reading the whole configuration file into memory does not make sense as Android devices do not have that much resources.

Luckily the algorithm for the checksum is very simple. Instead of writing in the file and calculating over the whole file, before each write a read is done to get the original value. The new checksum can be calculated by subtracting the two and adding the result to the current checksum. This way it is possible to keep the checksum up to date with each write.

4.2.2 ListViewAdapter

The UI is implemented in the ListViewAdapter class that uses the CpfManager class to get and set values into the configuration file. The ListViewAdapter class requests the CpfManager for a parser class object. It then uses the parser class to populate the UI with values from the configuration file and save changes that the user makes back to the configuration file.

Figure 4.7 shows the class diagram for handling the UI. In a general case the Cpf-Manager returns a parser object and the ListViewAdapter then generates a UI for it using the SeekBar and SwitchLayouts. Switch is used for bool values and Seekbar for the rest. This is done to satisfy a requirement for the application to be easily extendable. With this functionality for new IDs future developers only need to create a parser and add info about the ID into the MenuConstants.

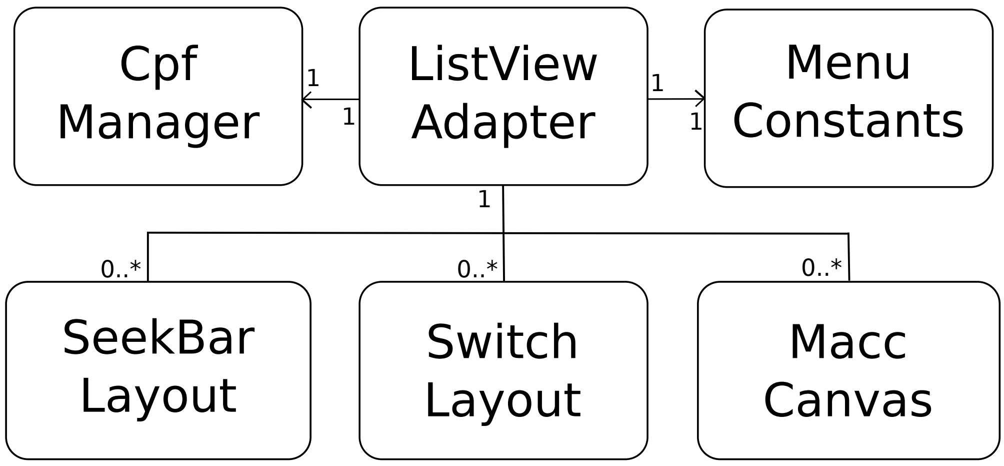


Figure 4.7 *Class diagram for handling the UI.*

Before using the generic UI the ListViewAdapter checks if there is a custom UI class for the ID in question. Currently only one custom UI is implemented and it is for the MACC. The configuration file can have three IDs that use this UI, one for still, video and the viewfinder.

4.2.3 MaccCanvas

The MaccCanvas class implements a custom UI element that allows the user to easily change the MACC data structure. The structure contains a array of 16 matrix elements that are two by two and contain 16 bit fixed point numbers. Each matrix corresponds to a section of the YCbCr color space. The ISP transform each pixel going through the pipeline with the matrix belonging to the pixels sector.

In the case presented in Figure 4.8 all the pixels that are in the section 1 of the YCbCr color space get moved by the red vector presented on the right side. The values for the black dots that the user manipulates go from -1 to 1 in both y and x dimensions. The points presented on the left side are the reference points that are used for the transformation. The first segments reference point coordinates are $(\frac{1}{2}, 0)$ the seconds $(\frac{1}{2}, \frac{1}{4})$ and so on. On the right side of Figure 4.8 the first segment's current value has been changed.

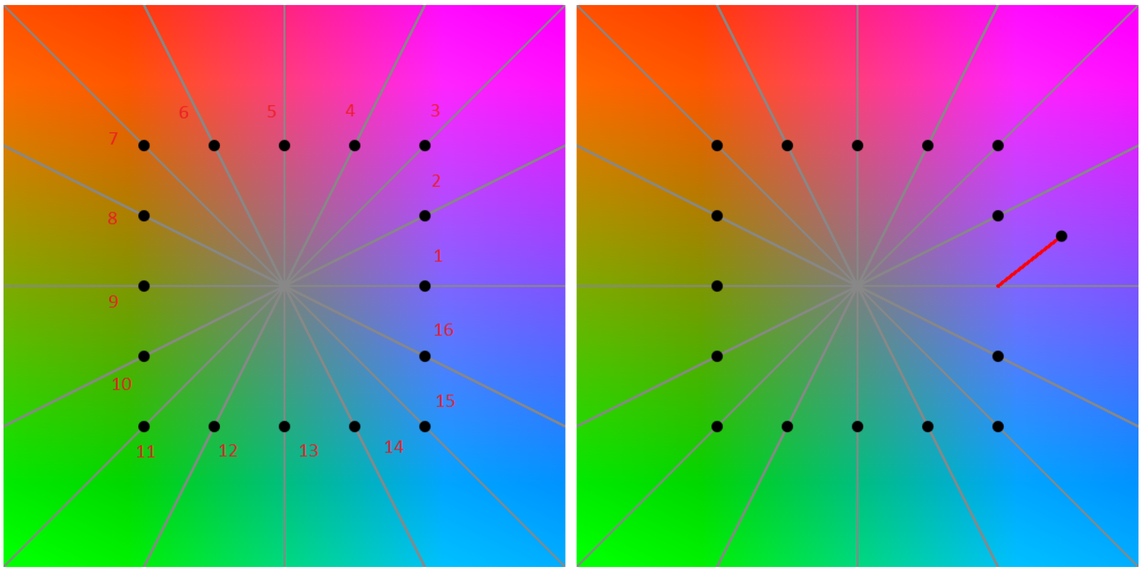


Figure 4.8 *The YCbCr colospace split into sections.*

Each matrix uses information from two of the segments, the current one and the one after it to eliminate discontinuity on the segment borders. Calculating each matrix in the configuration file (lets denote this matrix with C) from the points is done with a matrix division $C = B/A = B * A^{-1}$ where B and A are defined in equation (4.1).

$$A = \begin{bmatrix} rx_0 & rx_1 \\ ry_0 & ry_1 \end{bmatrix} B = \begin{bmatrix} cx_0 & cx_1 \\ cy_0 & cy_1 \end{bmatrix} \tag{4.1}$$

To calculate the division the inverse matrix for A is needed. Luckily the matrices are square so the inverse can be calculated with simple equation presented in (4.2).

$$A^{-1} = 1/det(A) * adj(A) \tag{4.2}$$

The determinant and adjugate for 2x2 matrix are also simple to calculate as presented in (4.3) and (4.4)

$$det(A) = a_{00} * a_{11} - a_{01} * a_{10} \tag{4.3}$$

$$adj(A) = \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix} \tag{4.4}$$

For the inverse operation if $C = B/A$ we get $B = C * A$ to calculate the actual coordinates from the matrices in the configuration file.

The points in the MaccCanvas are saved in xy-coordinates, but the tuning engineers want to change them in polar coordinates. This is because in polar coordinates the distance from orig corresponds to the colors saturation and the angle to the colors hue. To do this each time a point is changed we transfer the point to polar coordinates, add the wanted radius or angle and transfer back to xy-coordinates to save the new value.

The MaccCanvas class extends the Android View class that is the most basic UI element on Android. The onDraw method of the view is used to draw the MACC UI. The complete UI is presented in Figure 4.4. The canvas is populated by drawing the following items in order: a PNG image of the YCbCr color space, lines to separate the segments, the control areas on the edges, the lines from the reference point to the actual location and last the actual points (with a dot or circle depending if the point is selected or not).

4.3 Challenges

At the start of the project there were multiple challenges that needed to be solved before the project could actually begin production. The application needed to change a file under the Android file system's /system folder. The Android 4.4 update added new mandatory kernel feature called device-mapper-verity (dm-verity) verified boot [14]. The dm-verity system provides integrity checking of block devices and it is enabled by default for the /system folder preventing any changes.

The dm-verity can be disabled through ADB, but this needs to be done after each boot and doing this solely through the Android application proved to be impossible. This problem was solved by placing a symbolic link over the configuration file that points to the applications file system area where and through it the changes the application makes carry to the uploading driver. The installation of the application will thus require a script to be run that creates this link.

Another problem that was faced was updating the camera pipeline after the configuration file has been updated. The Android native camera library used to provide so called live tuning method in previous platform versions, but it had not been implemented in newer platforms which were used in our development. The live tuning was integrated into a development info dump function that is activated from the Android terminal when running the command `"dumpsys media.camera"`.

An alternative method was found that was to kill and restart the Android mediaserver process, but this method was much slower as it required to also restart the camera application. As it is important for the tuning engineer to be able to see changes that occur as close as possible, it was decided to file a bug to get the live tuning implemented in all current and future platforms.

4.4 Continuous Integration

Continuous Integration (CI) machine was set up that runs on a dedicated Windows PC. The machine uses Gradle to build the application which is the tool used by the Android Studio. Android Studio generates and keeps track of the build automation files so setting up the build environment for CI computer was done by installing a the build tools package provided by Google.

Along with the build tools a Jenkins tool was installed to the build PC. A commit to the mainline branch of the project will launch a project build along with clockwork analysis. There is also a possibility to add scripts to the project that will be run along with the compilation. These scripts can contain any tests the developers have created for the project. If the compilation succeeds and all analysis and tests pass the compiled binary will be committed into another repository that is accessible from a web UI.

The CI system was also important as it was known that the current developers are on a fixed term employment contracts. So when we leave the company there should be an easy way for anyone to get a build out and not having to set up their own environments.

4.5 Upcoming features

During the project three new use cases for the application were presented. These are still in development, but the initial ideas for the UI is presented in this section. First the tuning team would like to be able to take pictures in a way that would link the picture to the configuration file that was loaded at that time. Second they would like to be able to take pictures in fast succession with many different configuration files. Third use case was to change some tuning value in steps and take a picture in sequence with each step.

These use cases are for improving usability and also the productivity of the tuning engineer. They will not require any changes to the underlaying implementation of the configuration file parsing, but rather we will need to implement new UI features.

For the tuning application to have control of the images it was decided that a our team will implement our own camera interface using the Google's application program interface. This way the images appear within the applications memory and file system space. The naming and structuring of the saved images is still an open issue, but a first version of the UI is done.

Figure 4.9 presents our own camera interface which will implement the first use case. This feature is named "Single-capture mode". It has all the same controls embedded into it as the original system alert window and the user can switch between these two with the eye icon. In the bottom of the camera view there are three dots indicating the three use cases. User can switch between them by swiping the screen left and right which will produce a notification and the dot will move.

Figure 4.10 shows how the second use case is implemented that was named "Multi-capture mode". The user can select and unselect the configuration files they would like to take pictures with with a long press on the file. When the camera icon is pressed to take a picture, the application will cycle through each selected configuration file taking one picture once the file is loaded to the ISP.

When in multi-capture mode pressing the camera icon would cause the application to load the configuration file "PREPARED1.AIQB" into the ISP then take and save

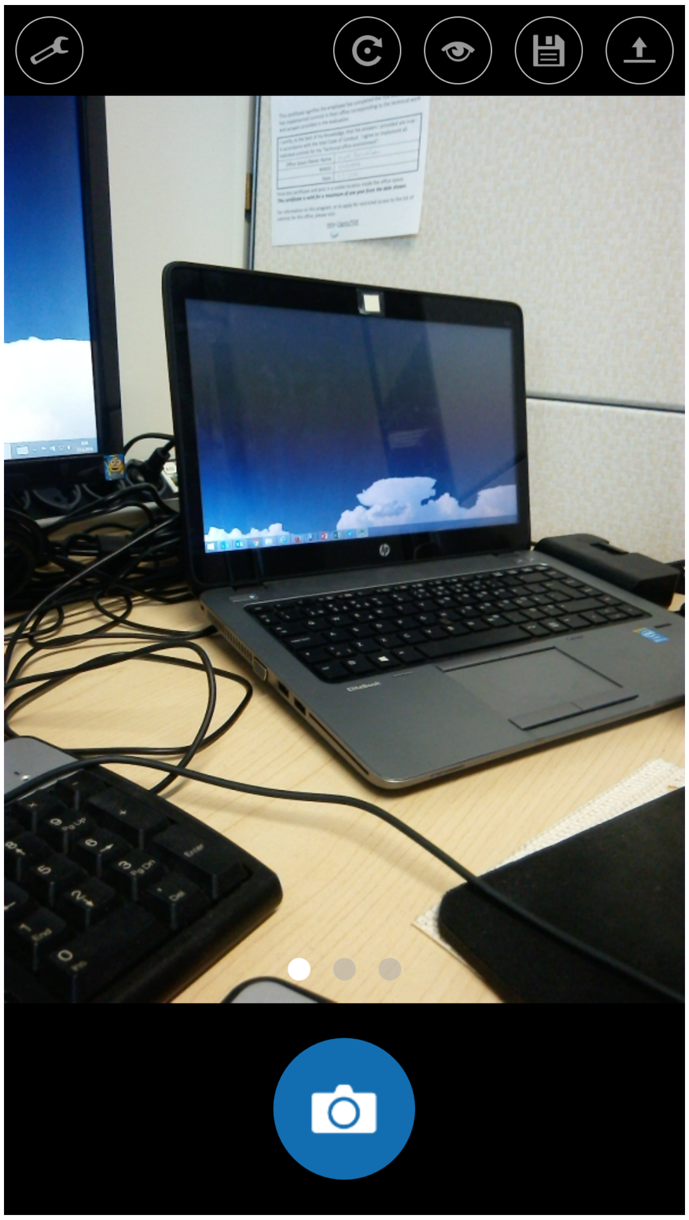


Figure 4.9 *Our own camera interface.*

a picture and iterate this process for the files “PREPARED3.AIQB” and “PREPARED4 .AIQB”. The tuning engineer would be expected to hold the device still as it takes 1-2 seconds to take each picture.

The point of the multi-capture mode is to let the tuning engineers prepare configuration files in advance and take pictures of one scene with each loaded into the ISP. Before this use case was implemented the way to achieve same results was to use multiple devices with each having different configuration file loaded through ADB.

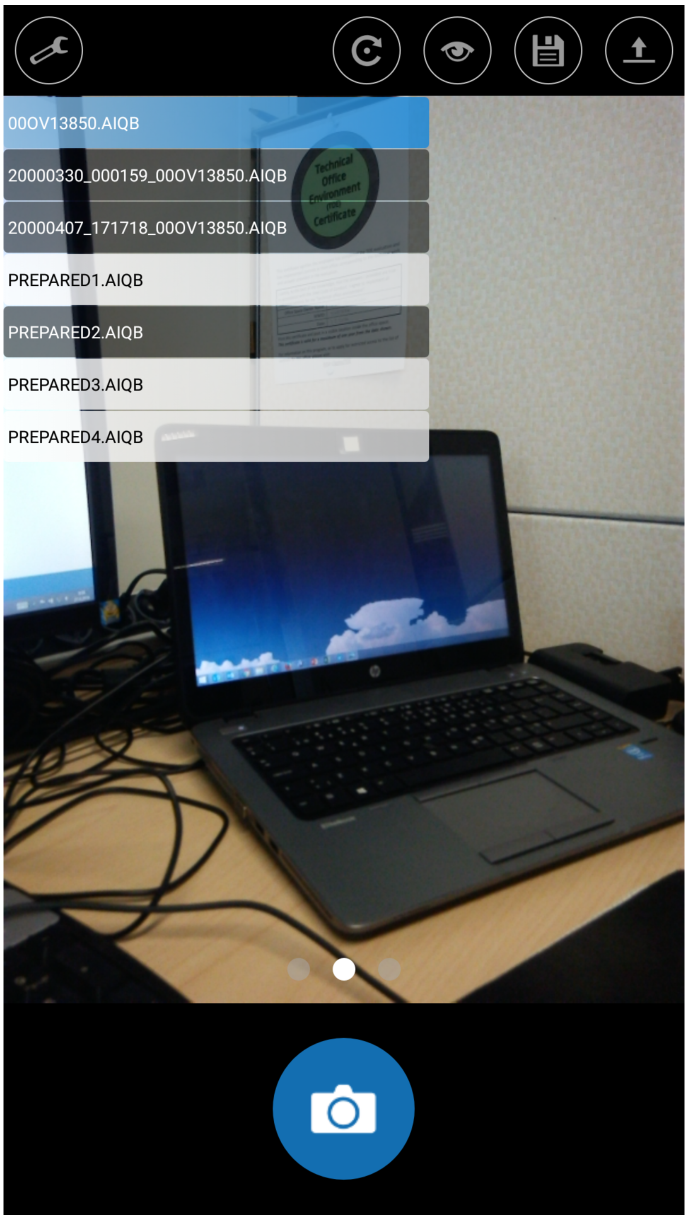
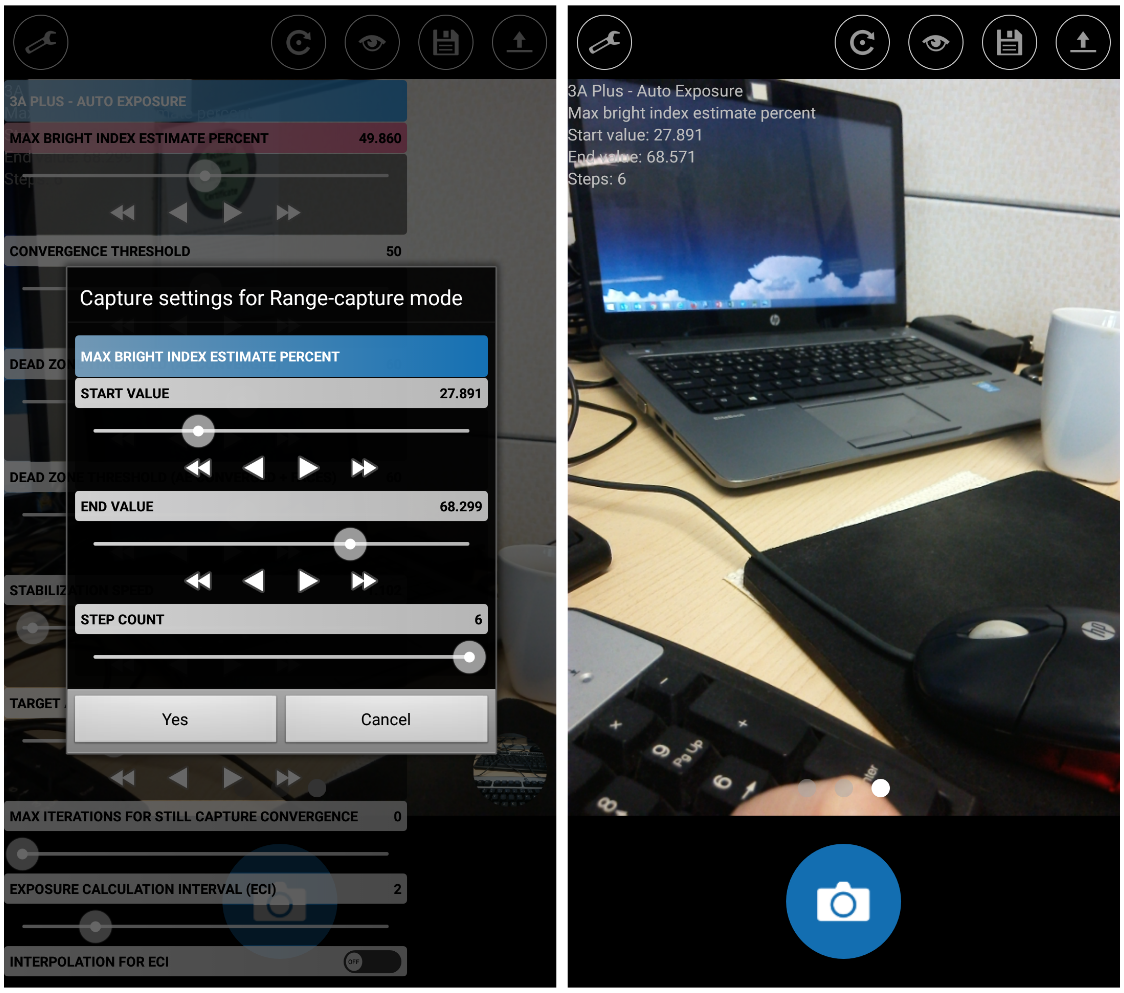


Figure 4.10 *Consecutive pictures with multiple configuration files.*

Figure 4.11 presents UI for the third use case named “Range-capture mode” . When in this mode the user can select any variable that can be changes via slider control. With a long press on the slider element the view presented on the left side of Figure 4.11 appears. Here the user can specify a range and the number of parts they want the range to be divided.

Once the range is specified the application will divide the range into steps and prepare a configuration file increasing parameter value for each step. Now the camera


3A PLUS - AUTO EXPOSURE

MAX BRIGHT INDEX ESTIMATE PERCENT 49.860

CONVERGENCE THRESHOLD 50

Capture settings for Range-capture mode

MAX BRIGHT INDEX ESTIMATE PERCENT

START VALUE 27.891

END VALUE 68.299

STEP COUNT 6

MAX ITERATIONS FOR STILL CAPTURE CONVERGENCE 0

EXPOSURE CALCULATION INTERVAL (ECI) 2

INTERPOLATION FOR ECI

3A Plus - Auto Exposure
Max bright index estimate percent
Start value: 27.891
End value: 68.571
Steps: 6

Figure 4.11 *Consecutive pictures within a given range.*

view will present the information of the selected range in the top left corner as seen in the right side of Figure 4.11. When the camera icon is pressed it will function similarly to the multi-capture mode, but use the prepared configuration files instead.

5. TUNING ALGORITHMS

The algorithms themselves are ultimately coded into the hardware and software of the ISP running inside the Android device. The algorithms take in variables from the tuning file. This way the same pipeline can be tuned to produce optimal image quality for any sensor. To use the application one must know the algorithm behind the variables to be able to do any meaningful changes to them.

The variables presented by the application can be split into two larger groups. Variables that control algorithms running on the ISP and the 3A+ algorithms that run on the CPU. The term 3A+ comes from the Auto Exposure (AE), Auto Focus (AF) and Auto White-balance (AWB). The plus at the end symbolizes other control algorithms that are grouped under the same term.

The ISP algorithms deal with pixels and always do the same operations specified by the tuning, such as color space transformation. This chapter will focus on the 3A+ algorithms to limit the scope and as the IQ team at the Finland site mainly focuses on those. The 3A+ algorithms run on the CPU and analyze the images in higher level and the control will happen for the next image the sensor will take. They can control the camera module and the some of the ISP parameters [15]. Figure 5.1 presents a block diagram on how the 3A+ algorithms control the image pipeline.

5.1 Auto Exposure

The AE algorithm uses histograms and percentiles to analyze brightness and contrast of the scene [16]. In photography histogram represents the tonal distribution of the image data over discrete intervals (bins). In a histogram graph the x-axis presents the dynamic range of the camera and is split into even sections called bins. The y-axis presents how many pixels in the image are within each bin.

Histograms are good for analyzing the exposure of images as they take the whole

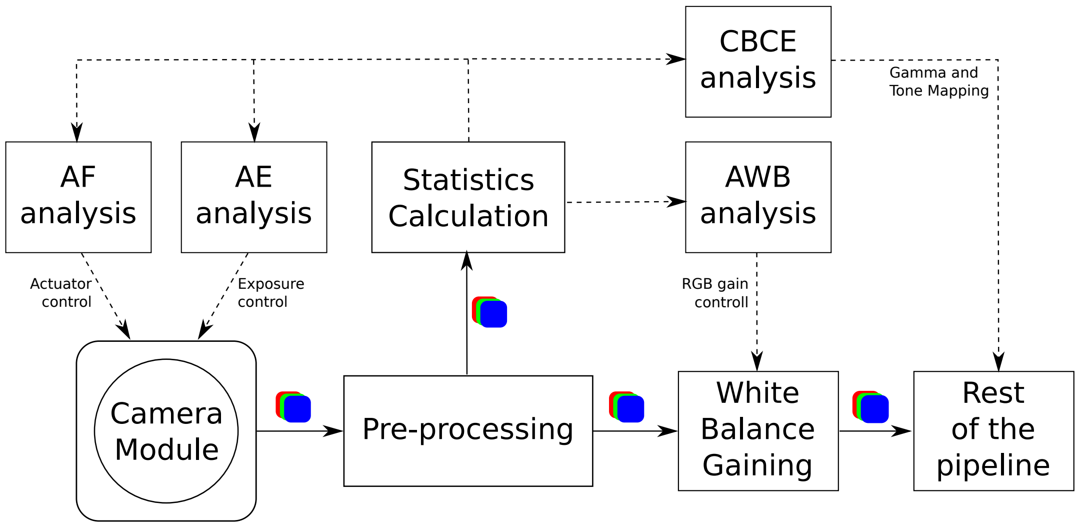


***Figure* 5.1** *3A+ algorithms in the ISP.*

scene into account instead of an average of the whole scene. Figure 5.2 presents one scene with different exposures. The scene contains shadows under the trees and bright lighting on the house and sky. In the over exposure the histogram has a high line on the right side. This is called clipping and means that some of the pixels are got fully saturated which means that no information in that part of the scene gets recorded. In the under exposure case the histogram is clumped to the left so the whole dynamic range of the camera is not in use. The correct exposure stretches the histogram over the whole dynamic range, but prevents clipping.

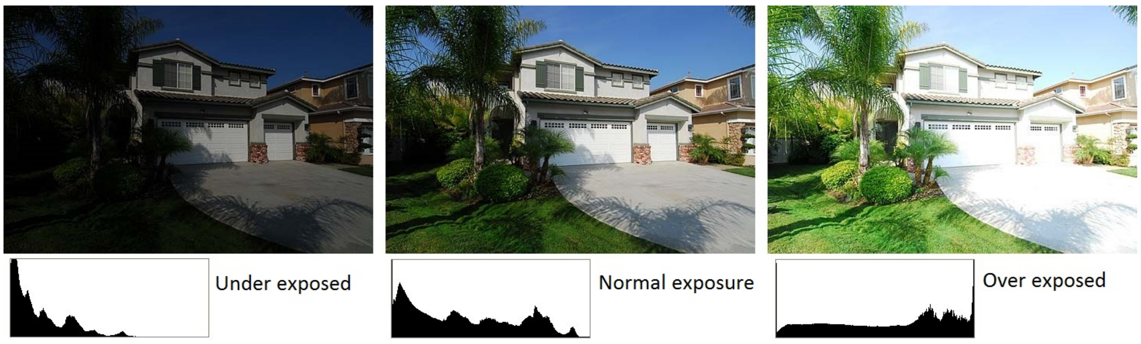


***Figure* 5.2** *Different exposures of same scene with histograms.*

The histogram can be calculated in different stages of the pipeline for example separately for each color channel or from combined R+G+B data. The AE algorithm uses two different histograms, a combined R+G+B histogram to prevent overexposure if one color channel is dominant and combined (R+G+B)/3 histogram to prevent underexposure if one color channel is locally saturated.

In statistics percentile is a value below which a some given percentage of observations in a group of observations will fall. The AE algorithm uses several different percentiles for the histogram values. These percentiles are presented in Table 5.1.

Table 5.1 *Percentiles in AE [16].*

Percentile	Usage
50th and 99.3rd	Percentiles for target average calculations.
5th, 10th, 90th and 95th	Percentiles for contrast estimation.
12th, 30th, 40th, 50th and 60th	Percentiles for black light compensation.

The AE algorithm first calculates the initial target brightness by stretching the histogram so that the current brightest data will move to the right end of the histogram so that 99.3% of all pixels can be found bellow this bin. This bright target is tunable and a default value is 85% of the dynamic range. This shifting is presented in Figure 5.3.

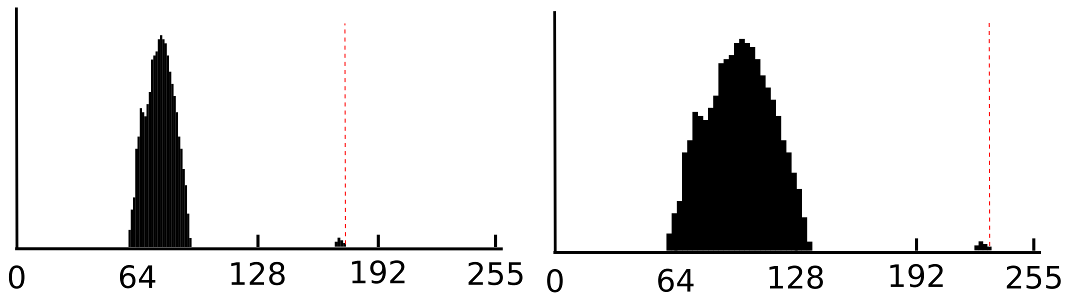


Figure 5.3 *AE initial target calculation*

After the initial target calculation AE may further stretch the histogram to the right using the 50th percentile also known as median. If the current median value is smaller than a tunable minimum median the histogram is stretched so that the minimum median is reached.

AE algorithm also implements several other algorithms for special situations such as back light compensation that detects human silhouettes under strong back light conditions and increases exposure time. Or face utilization where it prioritizes detected faces to make them exposed correctly. It is also possible to modify brightness according to user preference.

Distance from the convergence indicates how much image brightness differs from brightness where AE has its optimal target. The value range is [-1000‰, 1000‰] where the minimum and maximum values represent the length of the sensor's dynamic range. The AE algorithm may be converged even if the distance from convergence is far from 0 in cases such as dark scene where the sensor is unable to gather more light or if the maximum number of iterations have been used.

After the target from the histogram is found the exposure value is easy to calculate as the sensor is assumed to behave linearly. The target exposure is calculated with the formula in (5.1).

$$Target\ exposure = Current\ exposure \cdot \frac{Histogram\ target}{Histogram\ current} \tag{5.1}$$

In Table 5.2 all the variables for AE that are implemented in the application are explained. The Figure 5.4 gives a visual explanation for one of the variables in Table 5.2.

Table 5.2 *Variables for AE [17].*

Variable	Range	Explanation
Max bright index estimate percent	0.0 - 100.0	Multiplier for calculating the initial bright index.
Convergence threshold	0 - 100	Below this value, target average is stabilized with no additional weight (close to convergence).
Dead zone threshold	0 - 100	Dead zone threshold which is used after AE has converged (to avoid small changes in AE)
Dead zone threshold with faces	0 - 100	Dead zone threshold which is used after AEC has converged when face detection is used.
Stabilization speed	0.0 - 39.0	Speed of exposure time convergence.
Target average filter time	0.0 - 2.0	Target average stabilization time in seconds.
Max iterations for still capture convergence	0 - 10	Maximum number of preview iterations if AE has not converged and still capture is initiated.
Exposure Calculation Interval (ECI)	0 - 10	Number of frames to skip between AE iterations.
Interpolation for ECI	True/False	Interpolate the exposure values between each interval (see Figure 5.4)

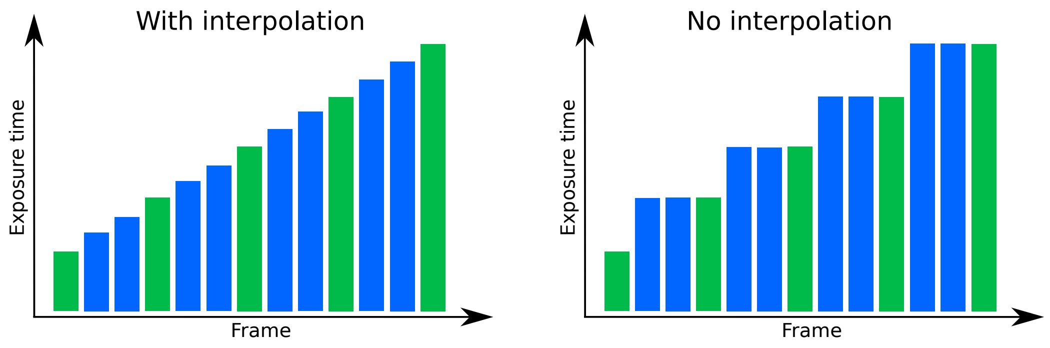


Figure 5.4 *Interpolation for ECI.*

5.2 Global Brightness and Contrast Enhancement

Global Brightness and Contrast Enhancement (CBCE) performs two operations, gamma adaptation and histogram stretching. By minimum CBCE needs to apply sRGB gamma adaptation to produce output images in sRGB color space. In a typical case the CBCE modifies the default sRGB gamma tone mapping in run-time to optimize the brightness and contrast of the particular scene [18].

Gamma correction is used to optimize bit usage in the image by taking advantage of the non-linear way the HVS preceives light and color. As seen in the Figure 5.7 gamma adjustment is non-linear and preserves more information than exposure adjustment that is linear mapping between input and output pixel values.

Figure 5.6 shows two ways to present the histogram stretching. Initial dark and bright offsets are calculated using percentiles and the “Dark target offset” and “Bright target offset” values in Table 5.3 may move these offsets further.

Once the bright and dark offsets are calculated the output of the CBCE algorithm is a stretched gamma table that will be applied to the image in the pipeline. In a simple case gamma correction is defined with equation presented in (5.2).

$$V_{out} = AV_{in}^{\gamma} \tag{5.2}$$

In a common case A=1 and both the input and output are in the range [0, 1]. For the gamma value γ common values fall in the range $[\frac{1}{8}, 8]$ [?]. Figure 5.7 presents

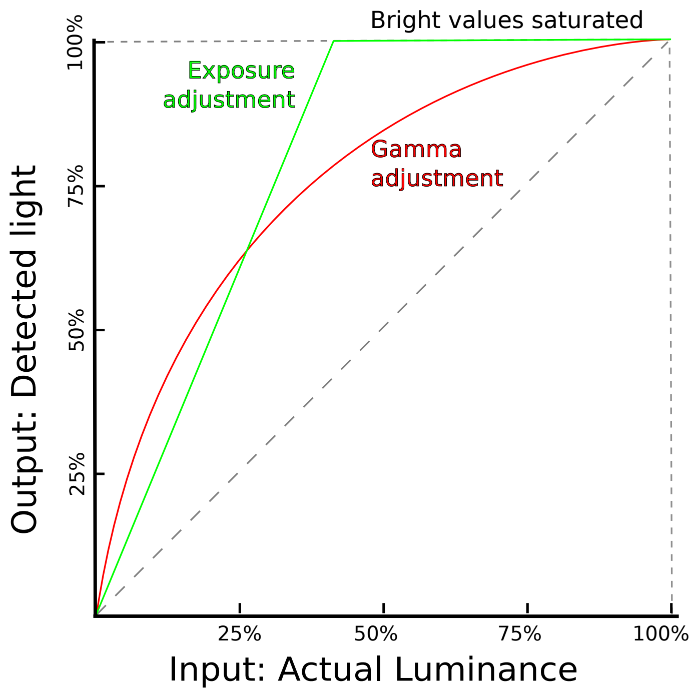


Figure 5.5 *Gamma correction.*

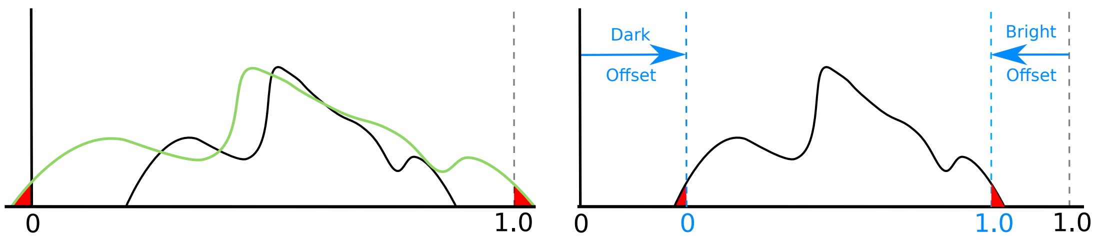


Figure 5.6 *Two ways to present histogram stretching.*

an image with different gamma corrections. But as CBCE uses a lookup table it is not restricted to setting the two values A and γ.

In Table 5.3 all the variables for CBCE that are implemented in the application are explained.

5.3 Auto White-balance

The AWB algorithm aims to locate the White Point (WP) which is used to correct the color discrepancy caused by the light source. Locating the correct WP is not an

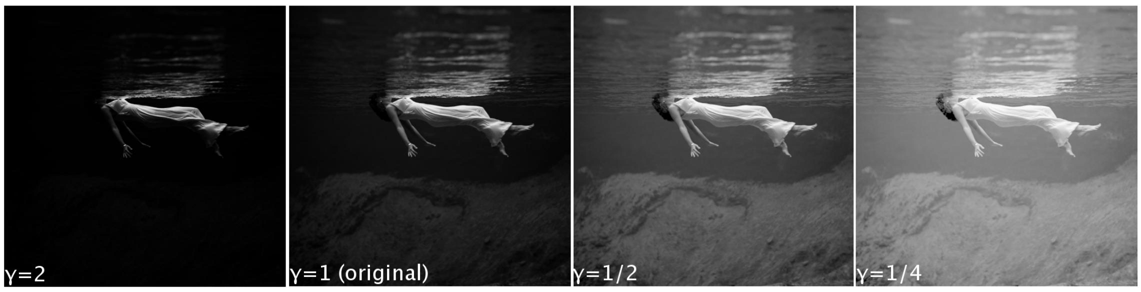


Figure 5.7 *Same image with different gamma values*

Table 5.3 *Variables for CBCE [17]*

Variable	Range	Explanation
CBCE level	0 - 2	GBCE "complexity" level enumerator
Bright target offset high contrast	0.0 - 1.0	Bright target offset in high contrast scenes
Bright target offset low contrast	0.0 - 1.0	Bright target offset in low contrast scenes
Minimum bright target	0.0 - 1.0	Minimum bright stretch percentage limit
Percentile dark target	0.0 - 1.0	Histogram percentile defining the initial dark target
Dark target offset	0.0 - 1.0	Dark target offset
Max dark target	0.0 - 1.0	Maximum dark stretch percentage limit

easy task as there are many different conditions and scenes that the same algorithm needs to work in [19].

Figure 5.8 presents three different scenes with increasing difficulty for the AWB algorithm. Easy scenes contain a lot of achromatic area or there are many different colors present. Difficult scenes have no large achromatic areas or only a few different colors. In difficult scenes the sensor response is ambiguous which means that an area could be chromatic or achromatic depending on illumination.

To find the WP the AWB algorithm uses many different algorithms with differing adaptive or fixed weights. Two examples of algorithms for locating the WP are the Grey World (GW) algorithm that assumes that the average chromaticity in an image to be gray (see Equation 5.3) or the gamut maximization algorithm that traces along the average chromaticity curve to find the WP which maximises the image color content upon correction.

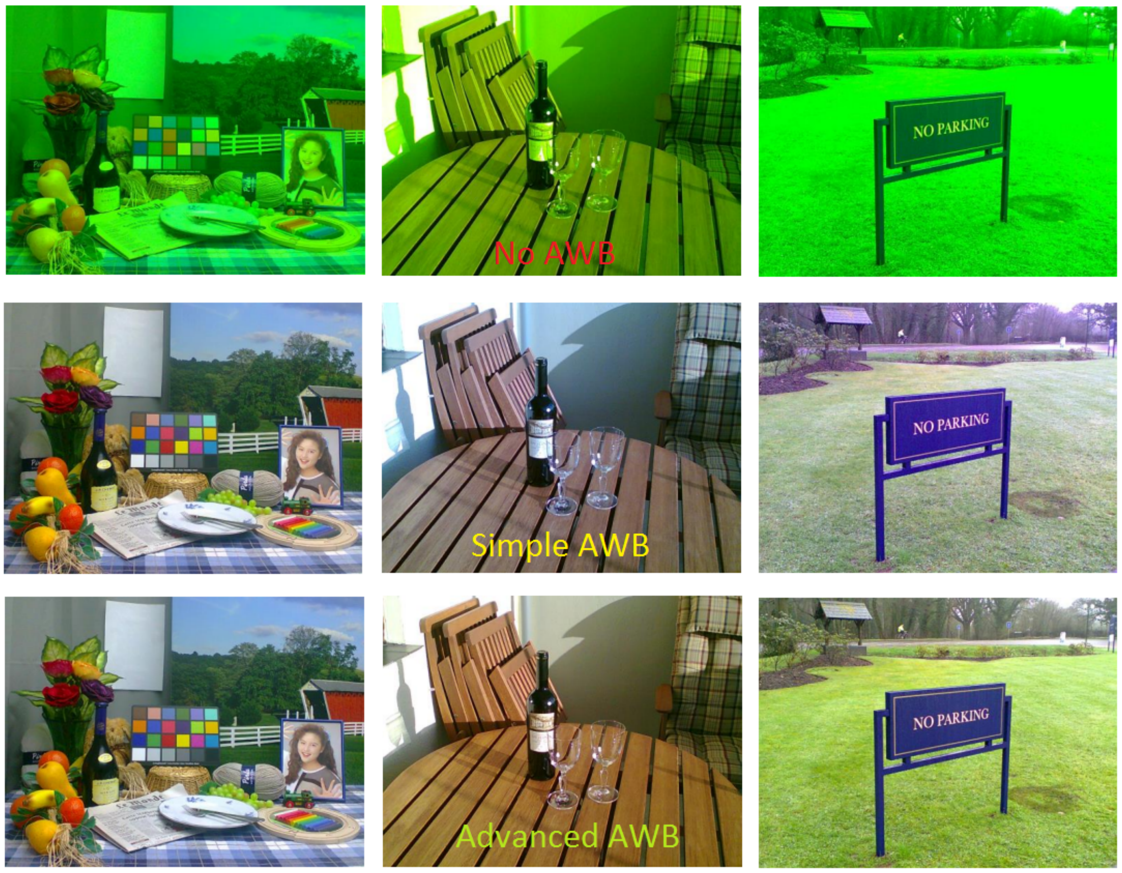


Figure 5.8 *AWB comparison [19].*

$$WP = \left[\frac{\sum\limits_{Pixels} R_i}{\sum\limits_{Pixels} G_i}, \frac{\sum\limits_{Pixels} B_i}{\sum\limits_{Pixels} G_i} \right] \tag{5.3}$$

Once the WP is located information from the characterization is used to calculate the color temperature of the scene. The line in Figure 5.9 presents the average chromaticity locus of the sensor. Chromaticity locus presents the color temperature of a black body radiator or in other words the light emitted by object that is heated so that it starts to emit light. Correlated Color Temperature (CCT) nodes are the output of the characterization. The AWB output CCT is computed by interpolating between the nodes CCT_n and CCT_{n+1} with weights dependent on the distance to the perpendicular lines to the average chromaticity locus.

In Table 5.4 all the variables for AWB that are implemented in the application are explained.

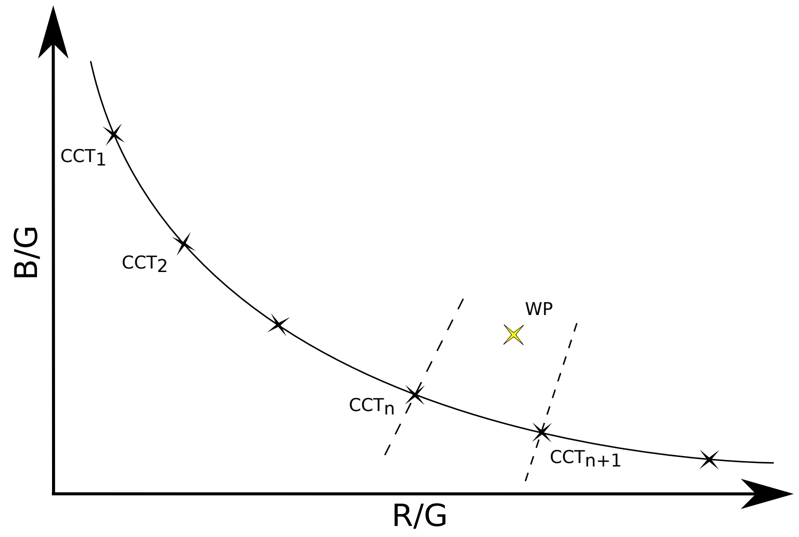


Figure 5.9 Average chromaticity locus.

Table 5.4 Variables for each CCT in AWB [17].

Variable	Range	Explanation
Convergence filter time for video	0.0 - 8.0	Convergence-filter time in seconds for video mode
Convergence filter time for preview	0.0 - 8.0	Convergence-filter time in seconds for video mode
Convergence filter time for Continious Capture (CC)	0.0 - 8.0	Convergence-filter time in seconds for CC mode
Chromaticity shift	1 - 30k	Linear-sRGB (R/G, B/G) chromaticity shift.
Constancy low	0.0 - 255.0	Color constancy corresponding to the lower interpolation boundary of the restricted CCT range

5.4 Auto Focus

Auto Focus (AF) aims to move the sensor optics in a way that the relevant part of the scene will stay in focus. Figure 5.10 presents different focus values for a same image.

The AF algorithm analyzes the contrast of the high pass filtered image content and comes up with a focus value. AF then uses the focus values to implement a hill-climbing search to find the global maximum[20].

Figure 5.11 presents the hill-climb search. The absolute values of the sharpness axis

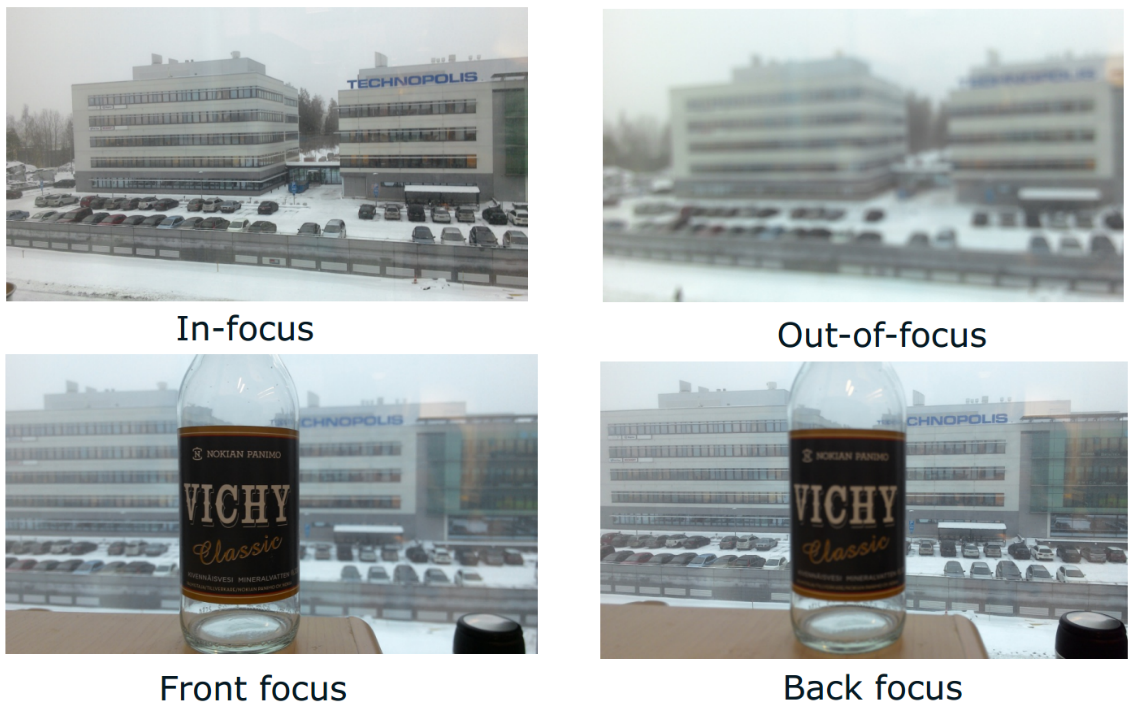


Figure 5.10 *Same image with different focus values [20].*

do not matter as the algorithm is only really interested in the relation between each step. The lens position values correspond to a voltage value that is fed into the lens actuator motor and depend on the camera module properties.

The AF implements a multi spot strategy where the best in-focus position is determined using multiple Window of Interest (WOI) regions in the center of the Field of View (FOW). The same hill-climb algorithm is run for each WOI and the best result is selected as the in-focus position.

In Table 5.5 all the variables for AF that are implemented in the application are explained.

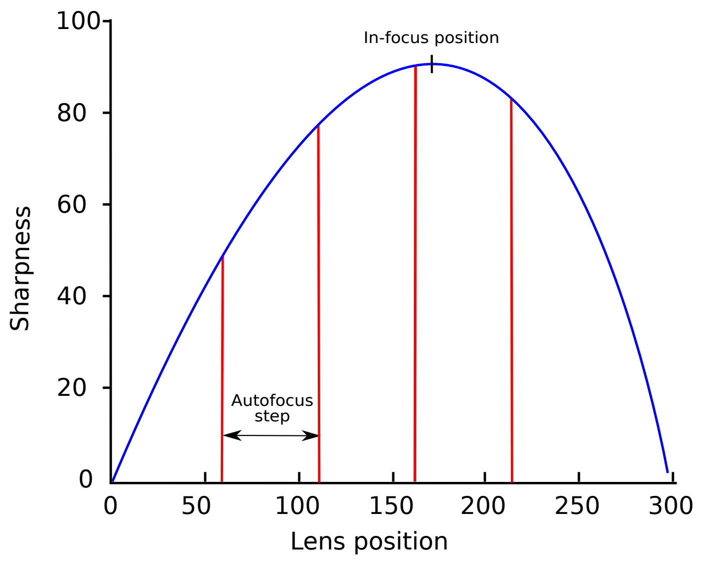


Figure 5.11 *Same image with different focus values.*

Table 5.5 *Variables for AF [17].*

Variable	Range	Explanation
Scene change multiplier	0.5 - 2.0	Multiplicative tuning parameter for scene change
Reference minimum contrast	0 - 200k	Minimum AF scene contrast magnitude
Search start still multiplier	0.5 - 2.0	Multiplicative tuning parameter for search start in still mode
Search start video multiplier	0.5 - 2.0	Multiplicative tuning parameter for search start in video mode
Min lux level suspend CAF video	0 - 100	Minimum illumination level for Continuous Auto Focus (CAF) in video
Max AF iter still	15 - 50	Max allowed number of AF iterations
CAF step scaling factor video	0 - 100	Multiplicative scaling factor for reference step size calculation in single video CAF
AF step scaling factor	192 - 384	Multiplicative scaling factor for reference step size calculation in single shot AF
Lux level AF assist auto	0 - 100	Lux level when AF assis light is triggered in flash auto mode
Lux level AF assist on	0 - 100	Lux level when AF assis light is triggered when flash is forced on
WOI number horizontal	0 - 10	Number of focus WOIs horizontally
WOI num vertical	0 - 10	Number of focus WOIs vertically
WOI width percent	0 - 100	Focus WOI width in percentage from the total frame width
WOI height percent	0 - 100	Focus WOI height in percentage from the total frame height

6. PROJECT PROGRESS AND RESULTS

The team consisted of two persons without any previous experience in Android application development. We started the project by doing tutorials on Android development with Android Studio and in the first month created a proof of concept application that was able to change the camera tuning configuration file. At this point we had solved most of the challenges and knew the application was technically possible for us to implement on Android. Figure 6.1 presents the lines of code in the Git repository during the project.

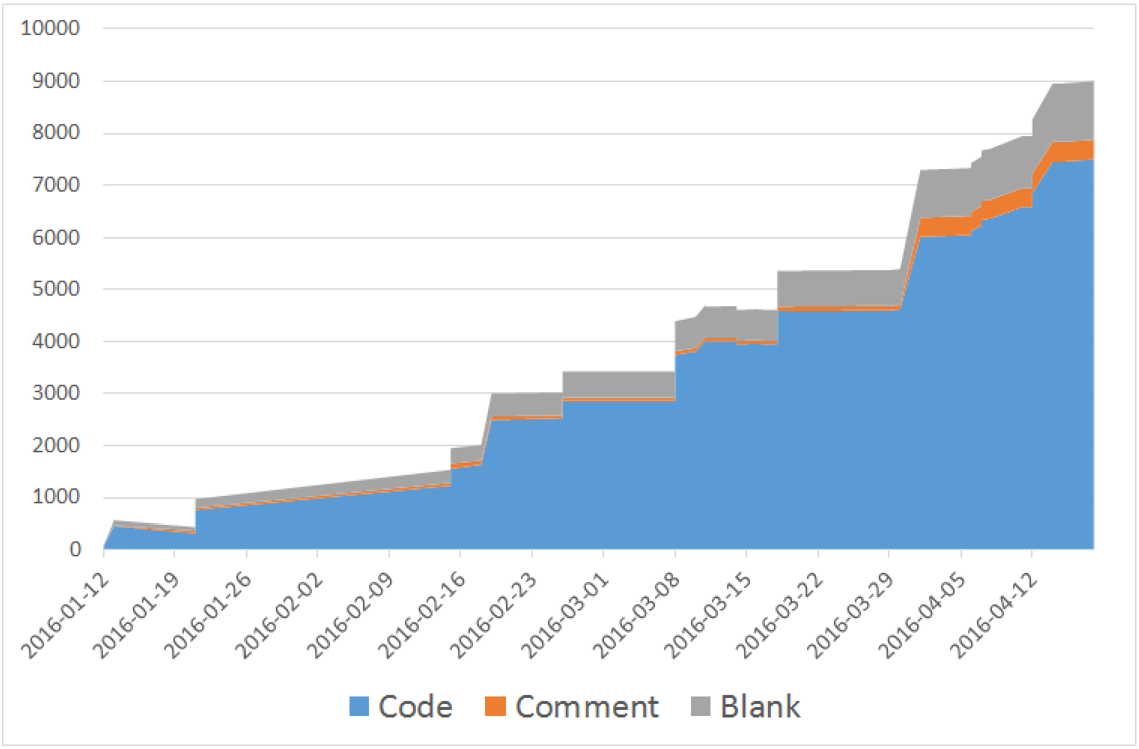


Figure 6.1 *Lines of code in the Git repository during the project.*

As the project was in a way a nice to have extra for the camera team where the manager was busy with other tasks, we were left to lead our selfs. At the start of the project we were a bit disoriented on how we should direct ourself. Luckily we were small two person team so it was easy to communicate on what we were doing.

We did a lot of pair programming so there was a lot of overlap in the responsibilities. A crude division of responsibilities was that I was responsible for the tuning file parsing and custom UI. I also did the low level fix for the camera driver to get the live tuning working. My partner was responsible of the general UI structure, how to present the menus in a easily expandable and intuitive way. Later he started working on the new requests from the IQ team and implemented the "Multi-capture mode" and "Range-capture mode".

We later put up a Kanban board up with post-its on the side of our cubicle which helped us to plan and more importantly it gave the manager easier visibility on the project. This has proved to be an effective way to lead the project with enough visibility and minimal overhead. Figure 6.2 is an image of the Kanban cubicle.

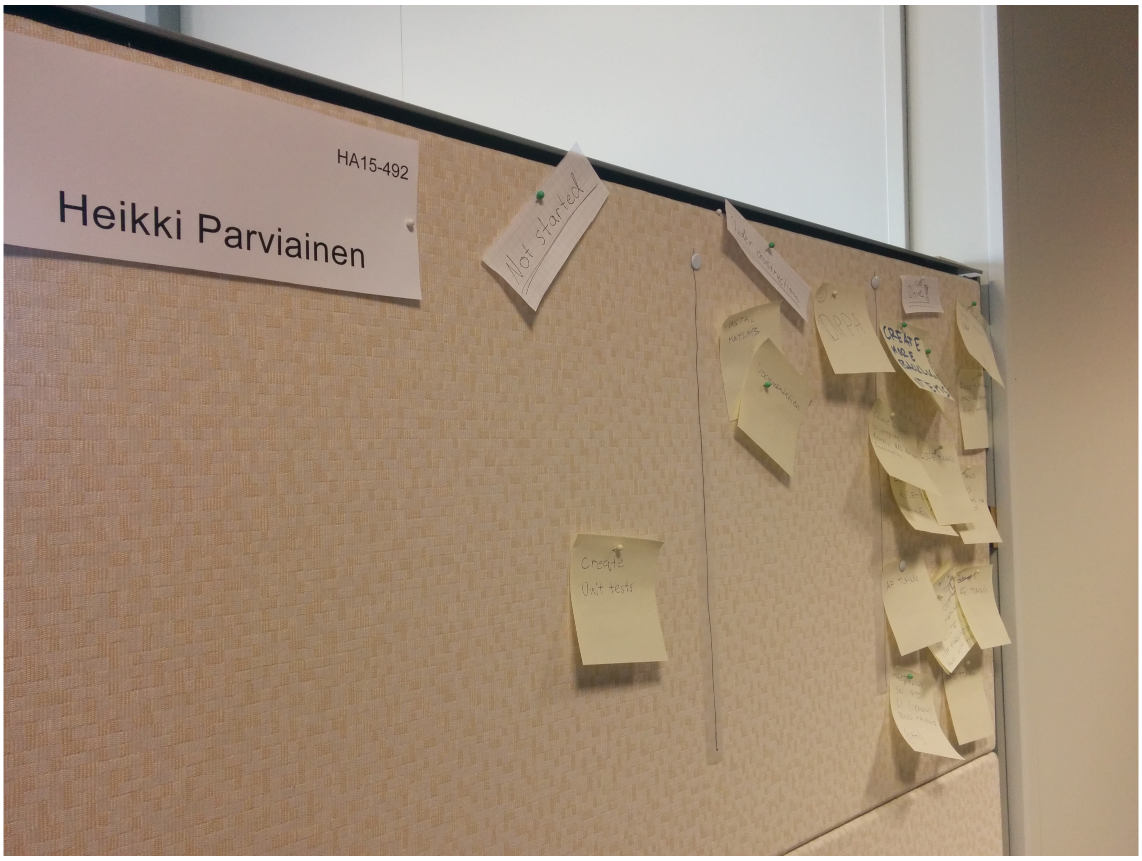


Figure 6.2 *Kanban board for the project.*

Getting requirements for the application was one issue during this project. We found out for both functional and UI requirements coming up with our own suggestion and presenting that gave out best results. Without setting the tone of the conversation with an example the answers tended to be too abstract to meaningfully implement.

Looking back on what was done the things that could have been done better are in designing the application and the testing. At the start we created the demo application and then started to expand on it. There was no intended design at the start. Later this was realized as the application grew and new use cases were too hard to implement. At this point we had a design meeting and re-implemented parts of the application based on the new design.

Implementing good unit tests was something that was done too late in the project. They would have helped much with finding issues with the parser classes. On the other side they were really laborious to create as each class needed its own test set.

Currently the application fulfills all the requirements we got in during the start of the project. It is able to do live tuning, it is easily extendable and we created a custom UI for the MACC ID which is similar to the one implemented on the IQ Tool. The tuning team also took interest on the application and we have received new use cases to implement. Some of the new use cases are displaying different color space in the MACC and taking consecutive images with multiple configuration files.

7. CONCLUSIONS

We met the original goal of the project that was to create a portable Android camera tuning application to help IQ team team to change the tuning file on the field. The tuning team was very happy about the application and a Intel recognition award for “Developing a tool that will significantly help IQ teams to make IQ tuning easier on fine tuning and field testing phase” was received for the work. The award is a way for the managers to acknowledge a work well done such as hitting a tight schedule or delivering some larger project successfully.

Other goal of the was to provide an application that is targeted for the customers using Intel’s System on Chips (SOC). The point is to offer the customer a simple and easy to use way to be part of the tuning process as the alternative is to use the IQ Tool that provides too many options for simple use. The Android application can be easily modified to present only a few simple parameters in the UI for this use case.

The application is still in development. The team will keep continuously improving it and adding new functionalities.

BIBLIOGRAPHY

[1] J. Nakamura, "Image Sensors and Signal Processing for Digital Still Cameras", 2006, Book, ISBN 0849335450

[2] Z. Li, T. Wei and R. Zheng, "Design of Black Level Calibration system for CMOS Image Sensor", 2010, Paper

[3] C. Chen, R. Hwang, Y, Chen, "A passive auto-focus camera control system", 2009, Paper

[4] N. Stauffer, "Active auto focus system improvement", 1983, Patent US 4367027 A

[5] http://masteryournikon.com/2013/01/31/what-is-white-balance/, Cited 2.3.2016, WWW

[6] R. Ramanath, W. Snyder, Y. Yoo, and M. Drew, "Color Image Processing Pipeline", 2005, Article

[7] P. Barten, "Contranst Sensitivity of the HUMAN EYE and its Effects on Image Quality", 1999, Book, ISBN 0819434965

[8] R. Lucac, "Demosaicked Image Postprocessing Using Local Color Ratios", 2004, Paper, 1051-8215/04 2004 IEEE

[9] W. Pratt,"DIGITAL IMAGE PROCESSING Digital Image Processing: PIKS Inside, Third Edition", 2001, Book, ISBN 9780471374077'

[10] K. Pulli, "Camera Processing Pipeline", 2015, Stanford lecture

[11] https://en.wikipedia.org/wiki/SRGB, Cited 12.3.2016, WWW

[12] https://photographylife.com/what-is-vignetting, Cited 20.3.2016, WWW

[13] D. Pascale, "RGB coordinates of the Macbeth ColorChecker", 2006, Paper

[14] https://source.android.com/security/verifiedboot/, Cited 30.3.2016, WWW

[15] T. Heinonen, "Fundamentals of Camera Tuning", 2015, Internal IQ Tuning course material

[16] M. Tuppurainen, J. Määttä, E. Krestyannikov, "Automatic Exposure Control", 2014, Internal training session

[17] Anon., "CPF 3A Plus Data", 2015, Internal datasheet

[18] B. Murat, "Global Brightness and Contrast Enhancement (GBCE)", 2015, Internal training session

[19] V. Uzunov, U. Tuna, J. Nikkanen "Automatic White Balance", 2014, Internal training session

[20] E. Krestyannikov, "Automatic focusing", 2014, Internal training session