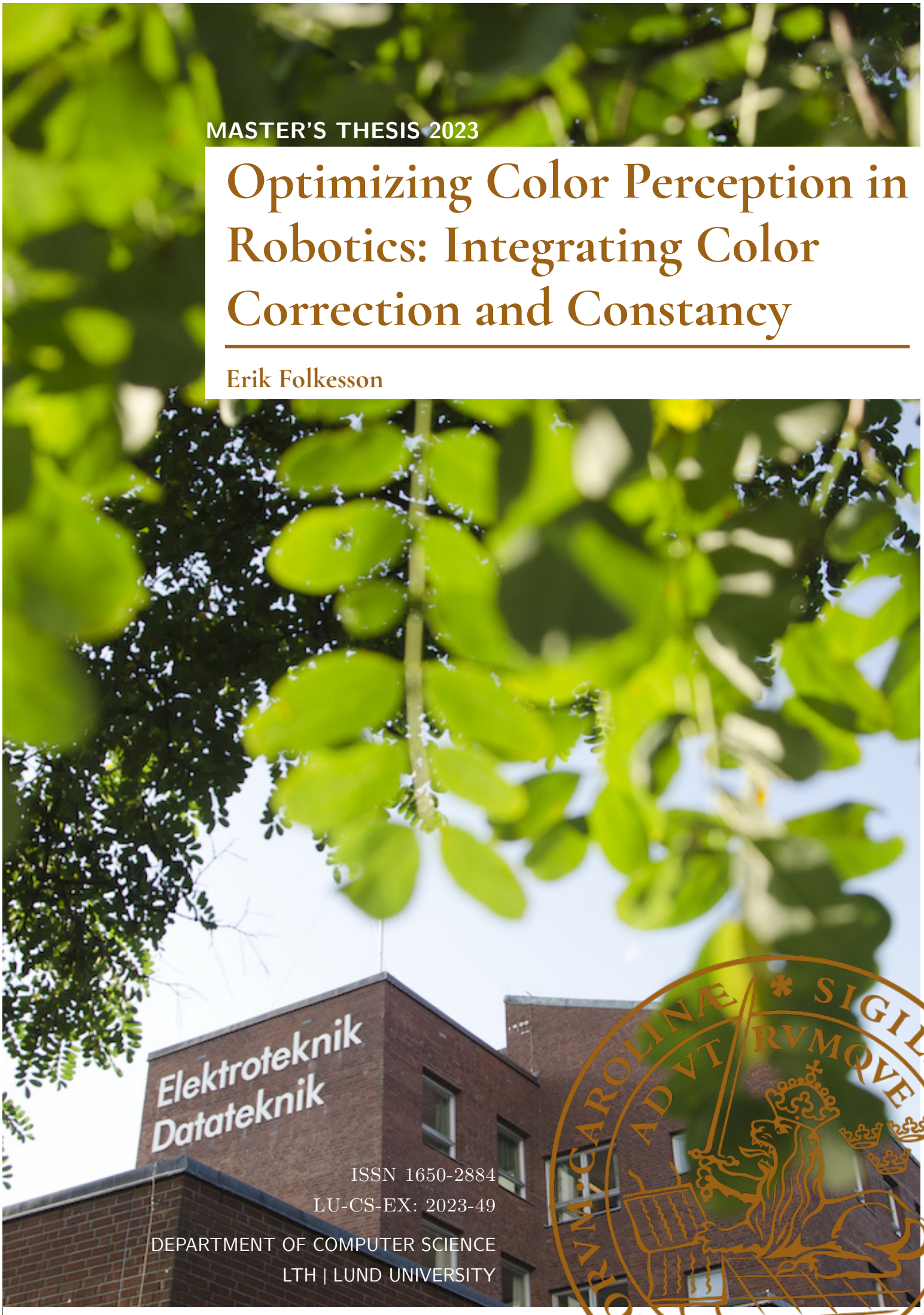


MASTER'S THESIS 2023

Optimizing Color Perception in Robotics: Integrating Color Correction and Constancy

Erik Folkesson

ISSN 1650-2884
LU-CS-EX: 2023-49

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-49

Optimizing Color Perception in Robotics: Integrating Color Correction and Constancy

Optimering av Färguppfattning inom Robotteknik: Integrering av Färgkorrigering och Färgbeständighet

Erik Folkesson

Optimizing Color Perception in Robotics: Integrating Color Correction and Constancy

(A LaTeX class)

Erik Folkesson
er7615fo-s@student.lu.se

December 8, 2023

Master's thesis work carried out at ABB Ltd.

Supervisor: Volker Krueger, volker.krueger@cs.lth.se
Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

This thesis explores the challenges and solutions associated with color perception in robotics, focusing on the integration of color correction and color constancy techniques. The study introduces a novel approach to color correction that does not rely on external tools, offering a plug-and-play method for color correction and color constancy in ROS2. The research investigates the viability of using a robot base as ground-truth over a color checker board for color correction. The potential improvement in color correction results through the use of color constancy is also assessed. The study also examines the effect of light source positions on any color correction reference used. The performance of various color correction and color constancy algorithms is evaluated in the context of robotics, with the best performing algorithms selected based on their closeness to a ground-truth image and their effectiveness in object detection under different lighting conditions. The findings provide guidelines for implementing color correction and color constancy in robotics. Although the novel approach hasn't yet outperformed conventional techniques in relation to object detection, the study suggests directions for continued exploration.

Keywords: MSc, Color Constancy, Computer Vision, Robotics, Color Correction

Acknowledgements

I would like to extend my thanks to my academic advisor, Volker Kreuger, for his guidance and support throughout the duration of this project.

Further thanks are due to my corporate advisors, Jonas Larsson and Aris Synodinos. Their professional insight and feedback have been integral to the completion of this thesis.

Appreciation is also expressed to ABB, the company for which this thesis was conducted. Their cooperation and provision of real-world applications for theoretical concepts have significantly contributed to this work.

Contents

Chapter 1
Introduction

Color, while a potent feature in computer vision applications, is a double-edged sword. Its susceptibility to changes in illumination conditions can significantly undermine the effectiveness of computer vision algorithms. Figure 1.1 demonstrates the substantial impact the illuminant's properties has on the final image captured by a digital camera. Neglecting this influence can lead to a significant degradation in the effectiveness of computer vision algorithms[18]. Our society's growing reliance on robotics across sectors, from healthcare to autonomous transportation, underscores the importance of accurate and consistent robotics. As we integrate robots more deeply into our daily lives and critical infrastructures, their ability to accurately perceive and interpret their surroundings becomes not just a technical challenge but a larger societal directive. Therefore, it is crucial to attempt to neutralize the influence of the illuminant, something that is addressed by the fields of *color correction* and *color constancy* [9] [34]. In this paper we will present to the reader what *color correction* and *color constancy* is and what differentiates them. Our investigation will focus on enhancing the reliability of color perception in robots that utilize computer vision applications, achieved by integrating color correction techniques with color constancy techniques. We will research existing color correction and color constancy algorithms with the aim of finding the best suited algorithm(s) for robotic applications. A select number of algorithms will be investigated and the resulting performance of these will be measured and presented.

In this project, our aim is to explore the implementation of color correction using external tools, such as a color checker (see Figure 1.2). Additionally, we will present and examine a novel approach to color correction that does not rely on external tools but instead utilizes the robot's surrounding environment. This makes the method more versatile and reduces the dependency on external calibration tools like color checker boards, which can be of extra importance in the context of mobile robotics. A comparative evaluation will be conducted to assess the efficiency of using external tools versus relying solely on the environment.

Moving from the theoretical to the practical, our final solution will be implemented in ROS2 Galactic[32], utilizing the Intel RealSense D405[14] depth camera and OpenCV for the computer vision components. The practicality and effectiveness of this implementation will

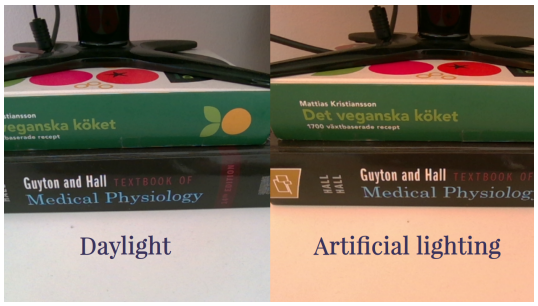


Figure 1.1: The Influence of Illumination on an Image (Note the color shift).

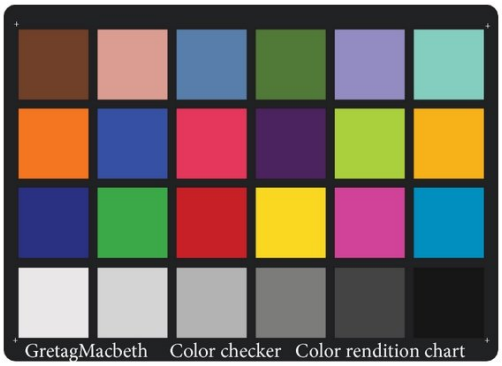


Figure 1.2: The MacBeth Color Checker[25].

be assessed in a real-world scenario involving the detection of colored blocks under diverse lighting conditions.

1.1 Project Scope

The scope of this project is to evaluate different color correction and color constancy algorithms in the context of robotics, specifically a robot arm with a camera mounted to it. The solution should require minimal calibration and setup time, should be easily portable to other robots and be able to handle a large variation in illuminants. We will be investigating two solutions to color correction, the first one using a color checker[25] board, such as the one seen in Figure 1.2 above. For the second solution, the color correction will be achieved by the novel approach of positioning the robot in such a way that the camera will be able to see the robot base, this will then be used as a consistent ground-truth point for executing color correction. We will also be investigating the use of color constancy as a way to further decrease the effect of the illuminant. A selection of color correction and color constancy algorithms will be implemented and evaluated. Here the performance of each algorithm will be measured and the best algorithm(s) will be selected. The effectiveness of each algorithm will be evaluated based on two primary criteria. Firstly, how accurately a calibrated image aligns with a ground-truth image will be measured using *rg-chromaticity MSE*, detailed in Section 3.4.2. Secondly, the efficiency of an object detection algorithm under varied lighting post-image calibration will be determined using a *Real-World Test*, discussed in Section 5.2.

1.2 Research Questions

In this project we have chosen to focus on the following questions:

- What is the viability of using a robot base as ground-truth over a color checker board for color correction?

- How does the integration of color constancy algorithms enhance the accuracy of color correction in robotics under dynamic lighting scenarios?
- How do varying positions of light sources impact the accuracy of color correction when using a robot base as a reference?

1.3 Contributions

This project contributes the following to the fields of color correction, color constancy and robotic computer vision:

- Introduces a novel approach to color correction that does not rely on external tools.
- An evaluation of existing color correction and color constancy algorithms in the context of robotics.
- Guidelines for implementing color correction and color constancy in robotics.

1.4 Outline

In the subsequent chapters we will delve into the background of color perception, discussing the role of color in image analysis and object recognition, and the digital representation of an image. A thorough explanation of color correction and color constancy is provided, including the mechanisms behind each process and the key factors that distinguish them from each other. The approach and implementation chapters introduce two solutions to color correction and investigate the use of color constancy to further decrease the effect of the illuminant. A selection of color correction and color constancy algorithms are implemented and evaluated.

We then move on to the evaluation of the implemented methods. The evaluation will be done first using a large dataset and then using a experimental setup that simulates a real-world scenario, namely object detection. The results chapter presents the findings from the dataset evaluation and real-world tests, and includes a discussion on the performance degradation in color correction algorithms, real-world testing results, and the impact of both color correction and color constancy in the setup. We conclude with a recap of the research problem and questions, a summary and interpretation of the findings, the limitations of the study, suggestions for future work, and the final conclusion.

Chapter 2

Background

This chapter begins by briefly discussing the role of color in computer vision applications, thereby emphasizing the significance of achieving consistent colors. We then delve into to the digital representation of an image, first exploring how an image is represented in memory, then explaining how a digital camera captures and creates these images. This understanding is fundamental to appreciating the following sections. Finally, we will provide a thorough and detailed explanation of color correction and color constancy. This will include an in-depth description of the mechanisms behind each process and a discussion on the key factors that distinguish them from each other.

2.1 Role of Color in Image Analysis and Object Recognition

We will now explore the practical applications of color in image analysis. One method that heavily relies on color information is *multispectral thresholding*. This technique involves applying intensity thresholding to each color channel in an image, as illustrated in Figure 2.1[13]. This approach can be used to perform edge detection in colored images[24], which in turn aids in the segmentation of an image into separate objects and areas. Such segmentation can facilitate object recognition in the image[16]. While there are numerous other applications of color in computer vision, such as the classification of objects according to color, the important point to note is that the use of color information generally enhances the performance of object detection and recognition[5]. Given its many applications and its capacity to enhance object detection and recognition, color stands as an indispensable tool in the realm of computer vision, serving as a cornerstone for the development of more accurate and efficient visual systems.

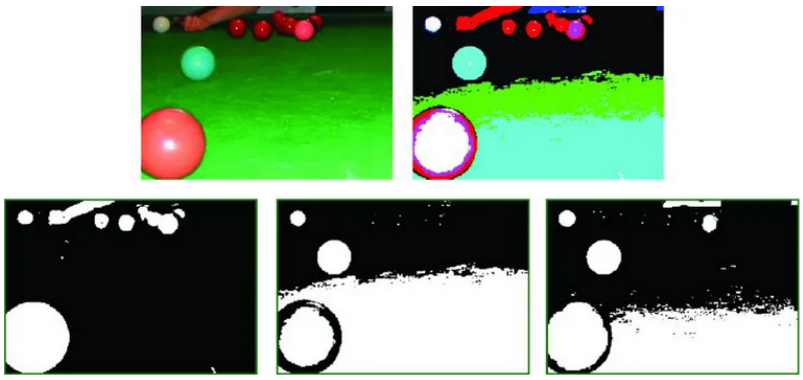


Figure 2.1: Multispectral Thresholding of an RGB Image[13]

2.2 Digital Color Image Representation and Image Formation

Now that we understand the importance of color in image analysis and object recognition, how exactly are these colors represented and formed in digital images?

2.2.1 RGB Color Image Representation

An RGB image is represented as a $m \times n \times 3$ matrix, where the '3' in the matrix denotes the depth corresponding to the three color components: red, green and blue. Each channel is represented by an 8-bit integer, meaning a value in the range $[0, 255]$. The color of a pixel in location (i, j) of the RGB image is determined by a combination of the red, green and blue intensities. This gives a potential of 16 million colors[23]. For an illustration of this representation, refer to Figure 2.2. Lets now look at how this representation is created, and what influences it.

2.2.2 Image Formation Model and the Role of Illumination in Color Images

The best way to illustrate how the image representation is created, and what influences this representation, is through the model presented below. This model will also help to create an understanding of why colors are inconsistent in images and why this problem does not have a perfect solution. Drawing from the work of Agarwal et al[2], the model has been reformulated and can be described as:

$$I_c(\mathbf{x}, \lambda) = \int_{\Omega} R(\mathbf{x}, \lambda) L(\lambda) S_c(\lambda) d\lambda, \tag{2.1}$$

where:

- c is a subscript that represents the sensor's response in each color channel, $c \in [R, G, B]$

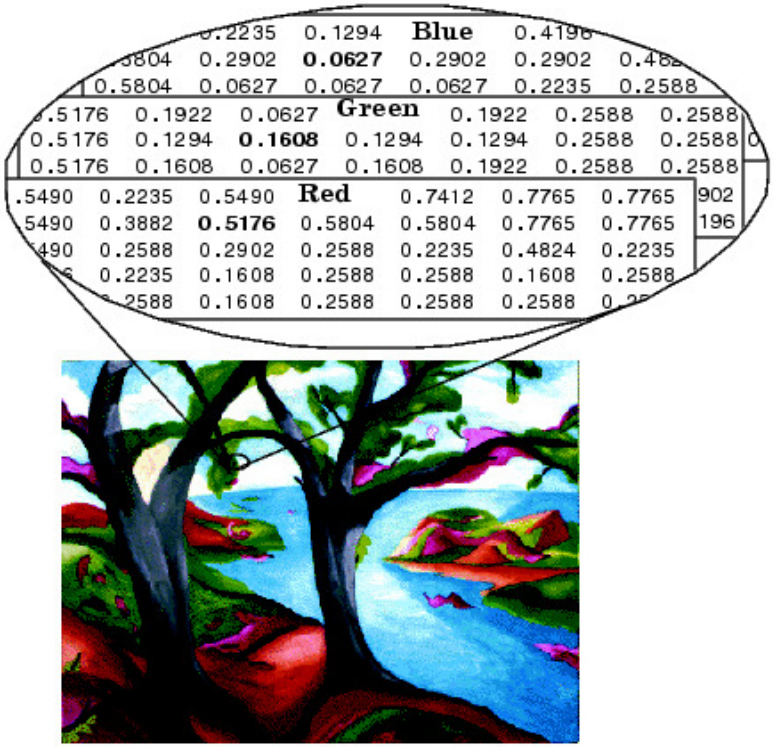


Figure 2.2: The Color Planes of an RGB Image[23]

- $I_c(\mathbf{x}, \lambda)$ is the sensor response corresponding to to the c^{th} channel. In other words, it's the intensity of the red, green, or blue color at a specific pixel location in the image. This intensity is determined by the interaction of the light with the object and the sensor characteristics.

- λ is the wavelength. Different wavelengths correspond to different colors in the visible spectrum.

- $\mathbf{x}$ is the 2D image coordinates. They specify the location of a pixel in the image.

- $R(\mathbf{x}, \lambda)$ is the surface reflection at a specific pixel location for a specific wavelength. It describes how the object's surface reflects light of different colors. This reflection depends on the material properties of the object.

- $L(\lambda)$ is the illumination property. It describes the properties of the light that is illuminating the scene. One important property is the color of the light. For example, sunlight has a different color than artificial light, which can cause the same object to appear as different colors under different lights.

- $S_c(\lambda)$ is the sensor characteristics for each color channel. Different sensors can respond differently to the same light, which can cause the same scene to be represented differently by different cameras.

- Ω is the visible spectrum. The integral over the visible spectrum takes into account all the colors of light that can potentially contribute to the image.

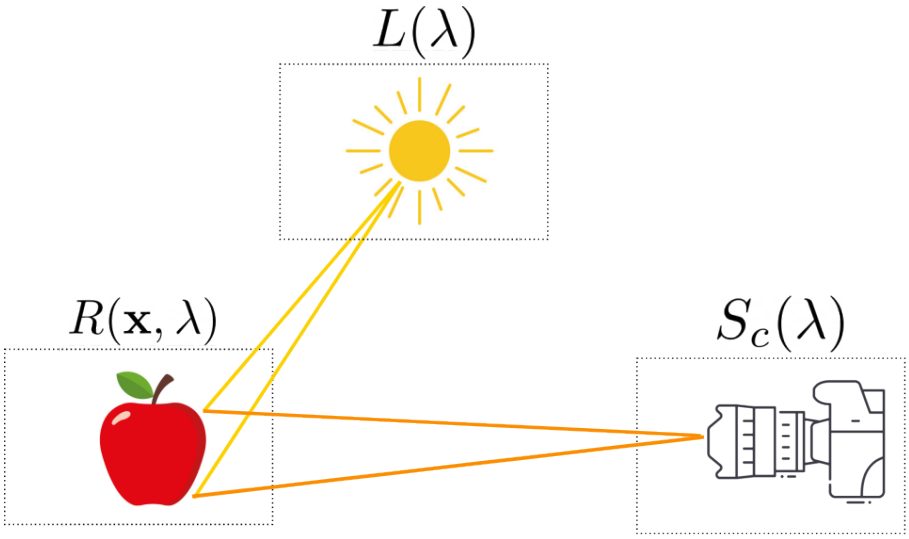


Figure 2.3: Illustration of the Image Formation Model in Digital Cameras

Figure 2.3 provides an illustration of this image formation model, helping to clarify the implications of the equation.

A closer look at Equation 2.1 and Figure 2.3 reveals that the final image is a product of the scene being captured ($R(\mathbf{x}, \lambda)$), the illuminant's properties ($L(\lambda)$), and the capturing device ($S(\lambda)$). In applications like object detection, the goal is to identify elements in the scene ($R(\mathbf{x}, \lambda)$). However, the scene's representation is influenced by both the illuminant ($L(\lambda)$) and the capturing device ($S(\lambda)$).

This implies that changes in illumination conditions or the capturing device can potentially degrade the performance of our object detection algorithm. Ideally, we would want an image that is solely dependent on the scene $R(\mathbf{x}, \lambda)$. This is precisely the objective of color correction and color constancy methods, which we will explore in the following sections.

2.3 Color Correction

Color correction is a technique that seeks to address the inconsistencies that arise from the image formation model, as outlined in Section 2.2.2. The goal of color correction is to transform an image, captured under specific illumination conditions and on a particular device, into a representation that is not dependent on either the illuminant or the device[9]. In the context of the image formation model (Equation 2.1), this implies eliminating the effects of both the illuminant, $L(\lambda)$, and the device, $S(\lambda)$, such that $L(\lambda) = S(\lambda) = 1$. Consequently, the equation can be reformulated as follows:

$$I_c(\mathbf{x}, \lambda) = \int_\Omega R(\mathbf{x}, \lambda) d\lambda, \tag{2.2}$$

This process involves converting the image from a representation that depends on the illuminant and the device (this image is commonly said to be in RGB-space) to an image that has a representation that is independent of both, as demonstrated in Equation 2.2 (this image

is commonly said to be in XYZ-space). This conversion is achieved through a mapping, as shown below:

$$\mathbf{q} = \mathbf{M}\rho, \tag{2.3}$$

where:

- ρ is a three element vector representing the three camera responses $[R, G, B]$. This corresponds to values found in the image $I_c(\mathbf{x}, \lambda)$ from equation 2.1.
- $\mathbf{q}$ is the corresponding device independent values, often called $[X, Y, Z]$.
- $\mathbf{M}$ is a 3×3 matrix that models the transformation. This is the mapping we aim to calculate.

To calculate the mapping $\mathbf{M}$, we need to select several $[R, G, B]$ values from the input image $I_c(\mathbf{x}, \lambda)$ and place them into the vector ρ. We also need an equal number of corresponding device-independent $[X, Y, Z]$ values to place in the vector $\mathbf{q}$. These device-independent values are derived from a ground-truth target, the nature of this ground-truth target and how the values can be obtained from it will be discussed in detail in Chapter 3.

Once we have the necessary values, we can begin to calculate the mapping $\mathbf{M}$.

However, one fundamental problem with this computation is the inherent need for an approximation, given that exact inverses for matrices $\mathbf{q}$ or ρ don't exist due to neither being square matrices. This has given rise to several methods for solving this equation, each with their unique strengths and weaknesses. We will introduce three such methods in the subsequent sections, these three methods are also the once that will be further investigated and evaluated later in this report.

2.3.1 Root-Polynomial Regression for Color Correction

Our first computational approach to determine the mapping matrix $\mathbf{M}$ is the *Color Correction using Root-Polynomial Regression*[19] approach, an enhancement of the conventional least-square regression technique, which we will be presenting below.

Least-Square Regression

Least-square regression is a method created by Finlayson et al[19] for calculating the mapping $\mathbf{M}$. The least-square regression method begins by defining a reflectance target matrix called $\mathbf{Q}$ and a corresponding camera response matrix called $\mathbf{R}$. Both these matrices are analogous to the vectors mentioned in the previous section, with $\mathbf{R}$ corresponding to the ρ vector that represents the three camera responses $[R, G, B]$, meaning the pixel values that are found in the image. The matrix $\mathbf{Q}$ corresponds to the $\mathbf{q}$ vector that signifies the corresponding device and illumination independent values $[X, Y, Z]$, meaning what the pixel values found in the image should be, if the effect of the illuminant and the device has been removed.

Using the Moore-Penrose Inverse[35] method, we can then compute the least-square of the following expression:

$$\mathbf{M} = \mathbf{Q}\mathbf{R}^{\mathbf{T}}(\mathbf{R}\mathbf{R}^{\mathbf{T}})^{-1} \tag{2.4}$$

Where:

- **R** signifies a $3 \times N$ matrix of camera responses, similar to ρ.
- **Q** symbolizes a $3 \times N$ matrix of the reflectance targets, comparable to **q**.
- **M** is a 3×3 matrix modeling the transformation.

While this computation does yield an approximate answer, it does not necessarily provide an exact answer which in our case means that we do not get a perfect mapping **M**.

Root-Polynomial Regression

The Root-Polynomial Regression method aims to augment the approximation's precision by using a polynomial regression of ρ. According to Finlayson et al[19] this means that we extend the vector ρ by adding additional polynomial terms of increasing degree. However, it's important to note that while this expansion may enhance performance under certain circumstances, it can also amplify the method's sensitivity to noise in the input data. The balance between accuracy and noise sensitivity will become more clear in Chapter 6, where we delve into the performance and limitations of this method.

2.3.2 Color Correction via Polynomial Term Expansion

Our second exploration in computing the transformation matrix is the *Polynomial Term Expansion*[9] method. To boost the method's efficiency, Cheung et al.[9] suggested the use of polynomial term expansion, in particular expanding ρ. For example, we can augment the ρ matrix to be an $8{\times}1$ matrix of expanded RGB values containing the terms $[1\, R\, G\, B\, R^2\, G^2\, B^2\, RGB]$. Accordingly, **M** then transforms into a 3×8 matrix.

The expression for the device and illuminant independent matrix **q** thus becomes:

$$\begin{aligned} X &= a + bR + cG + dB + eR^2 + fG^2 + gB^2 + hRGB \\ Y &= i + jR + kG + lB + mR^2 + nG^2 + oB^2 + pRGB \\ Z &= q + rR + sG + tB + uR^2 + vG^2 + wB^2 + xRGB \end{aligned} \tag{2.5}$$

By solving this polynomial, we derive the transformation matrix **M**. Although the accuracy of this method can potentially be improved by further expanding the polynomial's terms, it's important to note that increasing the number of terms also enhances the sensitivity to noise in the input data, mirroring the previous method's restrictions.

2.3.3 Color Correction using Vandermonde Matrix

The third and final method we explore for determining the transformation matrix **M**, involves the *Vandermonde Matrix*[36].

To define the Vandermonde Matrix let's consider a polynomial with unknown coefficients $\mathbf{A} = [a_0, a_1, \ldots, a_n]$ and known values $\mathbf{Y} = [y_0, y_1, \ldots, y_n]$. This polynomial can be

formulated as:

$$\begin{aligned} p(x_0) &= a_0 + a_1x_0 + a_2x_0^2 + \ldots + a_nx_0^n = y_0 \\ p(x_1) &= a_0 + a_1x_1 + a_2x_1^2 + \ldots + a_nx_1^n = y_1 \\ &\vdots \\ p(x_n) &= a_0 + a_nx_1 + a_2x_n^2 + \ldots + a_nx_n^n = y_n \end{aligned} \tag{2.6}$$

If we rewrite this in matrix form we get:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{2.7}$$

From this equation, we can directly define the Vandermonde matrix V as:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \tag{2.8}$$

By multiplying equation 2.7 with the inverse of the Vandermonde matrix, we can obtain the following expression:

$$VA = Y \implies A = V^{-1}Y \tag{2.9}$$

This expression, combined with polynomial degree expansion, as discussed in Section 2.3.2, is employed directly to solve for the transformation matrix $\mathbf{M}$. The reason this can be done is because the Vandermonde matrix is a square, non-singular matrix, meaning we can take the inverse of it. This method follows the same restrictions regarding noise as the two methods presented above.

With color correction presented, we will now move onto presenting color constancy.

2.4 Color Constancy

Color constancy aims to remove the effect of the illuminant $L(\lambda)$ and the device property $S(\lambda)$ from the final image $I(\mathbf{x}, \lambda)$[2]. Although it might seem similar to color correction, color constancy is distinctive in its reliance on assumptions rather than ground-truth targets.

The approach of color constancy involves making estimations to eliminate $L(\lambda)$ and $S(\lambda)$, based on a diverse array of assumptions. These assumptions widely vary in the color constancy literature, each carrying their own advantages and disadvantages based on what is being assumed. Thus, the effectiveness of color constancy is highly dependent on the applicability and accuracy of these assumptions in a given context.

In the sections that follow, we will delve into the five main categories of color constancy assumptions, presenting their respective advantages and drawbacks.

2.4.1 Different Categories of Color Constancy Algorithms

Hussain et al. [22] divide color constancy algorithms into five core categories, these are as follows:

Statistics-based Algorithms: function by making assumptions about image data, such as a particular average color intensity. A well-known example in this category is the gray world assumption, which assumes that the average of all the colors in an image should be gray. These algorithms can be simple and computationally efficient, but their performance may vary depending on the validity of their underlying assumptions.

Gamut-based Algorithms: operate by mapping the colors in an image (captured under an unknown illuminant) to the gamut of all colors achievable under a standard illuminant. Although these algorithms tend to be effective, they require foreknowledge of the standard illuminant, including its spectral power distribution — the energy profile across different wavelengths of light[17][33].

Physics-based Algorithms: use insights from the physics of light and its interaction with surfaces to calculate the illuminant properties[15]. For example, these algorithms might employ reflection models to approximate the illuminant properties[21]. Although these models can be highly accurate, their effectiveness often depends on the realism of the underlying physical models and assumptions.

Learning-based algorithms: rely on machine learning techniques. A prominent example of this approach is the use of Convolutional Neural Networks (CNNs) to carry out color constancy adjustments. These methods can be highly effective, especially with large and representative training datasets, but they may also be computationally intensive and potentially vulnerable to overfitting[22].

Biologically inspired methods: aim to replicate the human visual system and the way it perceives colors under various lighting conditions[37]. These methods can be fascinating and may offer insights into human perception, but their effectiveness in practical applications may vary.

In the next section we will be presenting a method where through combination of color constancy algorithms we can improve on the final illumination estimate.

2.4.2 Enhancing Illumination Estimate through Combination of Color Constancy Algorithms

Cardei et al.[6] demonstrated that combining different color constancy algorithms could yield significantly better and more consistent performance than any one algorithm could manage on its own. This improvement stems from the fact that all color constancy algorithms operate under certain assumptions about the world, which may be more or less accurate depending on the circumstances. By combining several algorithms, the error resulting from the failure of a single algorithm's assumption can be mitigated. Cardei et al.[6] proposed three

ways of integrating these various algorithms: a linear average weighting, an optimized linear weighting, and a nonlinear, machine learning-based weighting. They found that the optimized linear weighting produced the best results, and as such, this will be the combination method we examine further in Section 4.3.

Now that we've seen the benefits of combining multiple color constancy algorithms to optimize illumination estimates, let's turn our attention to the specifics of the three individual algorithms we've selected for further investigation.

2.4.3 Chosen Color Constancy Algorithms

In this section we will be presenting the three color constancy algorithm that we will be further investigating and presenting why they were chosen. After that we will be going into detail on each algorithm, detailing how they work.

The Gray World Algorithm: As one of the most widely used and recognized color constancy algorithms, the Gray World Algorithm offers a simple and efficient computational approach. This algorithm is an ideal choice for investigation due to its intuitive concept and wide applicability, making it a classic and fundamental method in the domain of color constancy[7].

The White Patch Algorithm: The White Patch Algorithm represents an effective solution for scenes with a dominant white color or bright areas. This algorithm operates under the premise that the brightest color in the scene should be interpreted as white, and the illuminant color can then be estimated from these areas. This assumption allows the algorithm to handle situations where the Gray World Algorithm might falter, making it a valuable complement to our study[7] [6].

Learning Based White Balance: With the rise of machine learning, algorithms that harness the power of large datasets have started to outperform traditional methods. Learning-based color constancy methods, such as Learning Based White Balance, can be highly effective especially with large and representative training datasets. Despite being potentially computationally intensive, they offer promising results and a way to adapt to diverse and complex real-world conditions. Therefore, they are crucial to consider in a comprehensive examination of color constancy techniques[8].

Gray World Algorithm

The *Gray World Algorithm* is based on the Gray World Assumption which says that the average color over an entire image should amount to gray. Any deviation from this norm is presumed to be caused by the influence of the illuminant. We can use this fact to achieve the primary goal of this algorithm, to remove the effect of the illuminant $L(\lambda)$. The first step in estimating the illuminant involves obtaining the average value of each color channel, which can done by using the following expression:

$$avg_c = mean(I_c), \tag{2.10}$$

where I_c are the individual color channels, $c \in [R, G, B]$, in the image I. The next step is to compute the light source intensity for each color channel, as follows:

$$L_c = \frac{2}{E(G)} avg_c. \tag{2.11}$$

In this equation, L_c represents the estimated illuminant for each color channel, while $E(G)$ is a geometry factor. Assuming the light source is perpendicular to the surface then we can simplify and use $E(G) = 1$[2] [7]. In the final step we combine our color channel illumination estimations, L_c, into a final estimation for L, giving us:

$$L = \begin{bmatrix} L_R & 0 & 0 \\ 0 & L_G & 0 \\ 0 & 0 & L_B \end{bmatrix} \tag{2.12}$$

This equation can be inverted and used in conjunction with Equation 2.1 to normalize $L(\lambda)$, which gives the following expression:

$$I_c(\mathbf{x}, \lambda) = \int_\Omega R(\mathbf{x}, \lambda) L^{-1}(\lambda) L(\lambda) S_c(\lambda) d\lambda = \int_\Omega R(\mathbf{x}, \lambda) S_c(\lambda) d\lambda. \tag{2.13}$$

The expression above shows that the effect of the illuminant has been removed from the final image, thus helping us achieve the desired color constancy in the image. [7]

White Patch Algorithm

The *White Patch Algorithm* is the second color constancy method we are investigating. Much like the Gray World Algorithm, it operates based on assumptions about the image content. Specifically, it assumes that the highest value in each color channel signifies the representation of white in the image.

The illumination estimation process in this algorithm mirrors that of the Gray World Algorithm, with the only difference being that it uses the maximum value instead of the mean value. This can be expressed in the following way:

$$max_c = max(I_c), \tag{2.14}$$

where, I_c represents the individual color channels, with $c \in [R, G, B]$, in the image I.

The algorithm assumes that the illuminant for each color channel corresponds to the maximum value of that channel, resulting in the following straightforward equation:

$$L_c = max_c. \tag{2.15}$$

We combine these into a final illumination estimation in the same way as seen in Equation 2.12 [6] [7].

Learning-Based White Balance

The third and final algorithm under consideration is a *Learning-Based White Balance* method, proposed by Cheng et al[8]. This approach utilises image features and a bank of regression

trees to estimate the illuminant. The features are all extracted while in *rg-chromaticity space*[8], a representation that decouples colour information from its luminance[10], with luminance indicating the amount of light emitted from, passing through, or reflected off an object[28]. The rg-chromaticity-space is computed as follows:

$$\begin{aligned} r &= \frac{R}{(R+G+B)} \\ g &= \frac{G}{(R+G+B)} \end{aligned} \tag{2.16}$$

Four features are leveraged as input for the bank of regression trees: *average color chromaticity*, *brightest pixel chromaticity*, *dominant color chromaticity*, and *chromaticity mode of the color palette*. Each of these features are described below.

Average color chromaticity is the chromaticity, (avg_r, avg_g), of the average RGB value in the image. The average RGB value can be calculated from Equation 2.10, which will give us the RGB values (avg_R, avg_G, avg_B). We can then transform this to rg-chromaticity space using Equation 2.18. It is worth noting that this is equivalent to the input to the Grey World Algorithm.

Brightest pixel chromaticity is the chromaticity, (max_r, max_g), of the brightest RGB value in the image. The brightest RGB value can be calculated using Equation 2.14, which gives us the RGB values of (max_R, max_G, max_B). Same as above we then transform this value into rg-chromaticity space using Equation 2.18. Here, it is interesting to note that this is equivalent to the input to the White Patch Algorithm.

Dominant colour chromaticity is the chromaticity of the average RGB value, belonging to the colour histogram bin with the highest count. In the original paper 128 bins were used per color channel, meaning a total of 128^3 bins were used. What this means is that we take each color channel and divide the values of that channel into N bins. Then, for each color channel histogram we find the average value of the bin with the highest count. This gives us a set of RGB values (dom_R, dom_G, dom_B). As before, we transform these RGB values into the rg chromaticity space, giving us (dom_r, dom_g).

Chromaticity mode of the colour palette represents the average value of each bin in the RGB histogram previously mentioned that contains more than a minimum number of values. In the original paper, this threshold was set to 200 values, which gave a total of roughly 300 different colours for an average image. Taking the average of these 300 colours, the RGB values $(mode_R, mode_G, mode_B)$ are produced. We then transform these values into the rg-chromaticity space, giving $(mode_r, mode_g)$.

These four features now need to be used to give a prediction on the illuminant. Here a learning based method is used. This method is based on *variance reduction regression trees*[27]. For each feature a series of K regression trees are estimated. The original paper estimated the regression trees in pairs, one regression tree for the r-chromaticity and one for the g-chromaticity. Each regression tree pair receives one feature at a time as input, which means that for each feature two regression trees need to be computed, the r and the g chromaticity trees. This means that for each feature $2 * K$ different illumination estimates are generated. In total we will end up with $8 * K$ illumination estimates as we have four features. When two

or more trees give similar illumination estimates these results are saved in two output sets, $\mathcal{R}$ for the r estimation and $\mathcal{G}$ for the g estimation. The final rg-chromaticity estimation of the illuminant is given by:

$$L_{r,g} = (mean(\mathcal{R}), mean(\mathcal{G})) \tag{2.17}$$

We can then apply the inverse of $L_{r,g}$ to the image while still in rg-chromaticity space[8]. We then convert the image back to RGB space though the following equations:

$$\begin{aligned} R &= \frac{rG}{g} \\ G &= G \\ B &= \frac{(1-r-g)G}{g} \end{aligned} \tag{2.18}$$

where G is the original intensity of the green value for that pixel. This is needed as rg-chromaticity does not encode intensity.

2.4.4 Example of Color Constancy - The Gray World Algorithm

The gray world algorithms is based on the the gray world assumption that says that the color of each sensor averages to gray over the entire image. Meaning that the average red intensity is equal to the average green intensity and the average blue intensity, this can be expressed as $Red_{mean} = Green_{mean} = Blue_{mean}$. This assumption holds best when there are a lot of colors in the image. This can be used to estimate the illuminant by taking the mean of each color channel and then looking at the deviation from this assumption, as discussed in Section 2.4.3[7]. As can be seen in Figure 2.4 the input image has a very large $Green$ intensity while the other two color channels have small values. In the output image this variation has been greatly reduced and the image is drastically less green. However, it is easy to see the gray world assumption does not hold if we have a large, single colored object in the scene. For an illustration of this problem see Figure 2.5. Here, the gray world algorithm made the image drastically worse because it will always assume that the average color should be gray, no matter what the content of the image is.

As demonstrated by the last example, the gray world example suffers when the scene contains a dominant color. To counteract this effect we can combine multiple different color constancy algorithm. This has the effect that when one algorithm leads us astray, the other algorithm can hopefully lead us closer to a correct response.

In the next chapter, we will delve into a comprehensive discussion on how color correction and color constancy can be integrated into one cohesive pipeline. We will be giving a concrete example of the effect color correction has on the image and based on this we will be presenting our innovative approach to the color correction problem. We will also be presenting a proposed workflow to effectively process images in the context of robotic vision application.

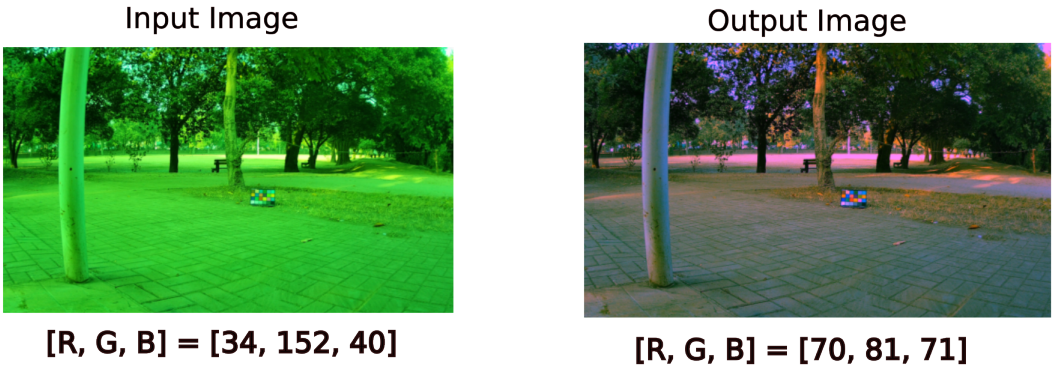


Figure 2.4: Effect of Gray World Algorithm + Average RGB of Image. Modified image from 10xEngineers[4]

Figure 2.5: Gray World Assumption on raspberry's. Modified image from Wikipedia[31]

Chapter 3

Approach

In this chapter, we will provide a thorough explanation of the methodology used to tackle our research questions. Initially, we will explore the conventional color correction technique that involves a color checker, demonstrating its impact on an image through a practical example. This discussion will set the stage for the introduction of our innovative approach. To underscore the practicality and effectiveness of our method, we will provide an in-depth demonstration of its effect on the image. We will then outline how our approach integrates into the workflow of a robotic vision application. The chapter will conclude with a presentation of our plan for evaluating our approach's performance.

Before we proceed, let's revisit the image formation model, Equation 2.1:

$$I_c(\mathbf{x}, \lambda) = \int_{\Omega} R(\mathbf{x}, \lambda) L(\lambda) S_c(\lambda) d\lambda. \tag{3.1}$$

This equation will serve as the foundation for understanding how the methods presented in this chapter modify the image. Now, let's delve into the most common method for executing color correction, using a color checker.

3.1 Color Checker Based Color Correction (CCB-CC)

The conventional approach to color correction is by using a color checker. We will be demonstrating the inner workings of Color Checker Based Color Correction (CCB-CC) through the example that follows.

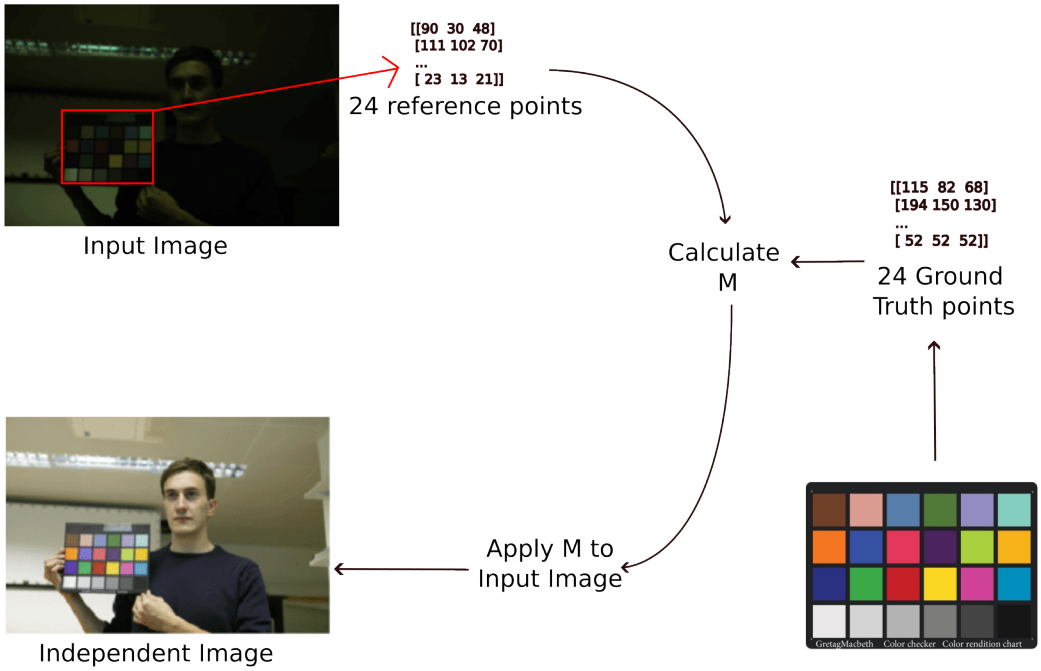


Figure 3.1: Flowchart of Color Correction using Color Checker. Image adapted from Figure 3, Lai et al[26].

3.1.1 Example of Color Checker Based Color Correction (CCB-CC)

This example will illustrate the effect that CCB-CC has on the final representation of the image. In this example we have a scene, $R_{in}(\mathbf{x}, \lambda)$, that we assume contains a color checker, an example of a scene that fulfils this assumption can be seen in Figure 3.1. The scene is captured by a camera with the properties $S_{in}(\lambda)$ under an illuminant with the properties $L_{in}(\lambda)$. This gives us an input image I_{in} that can be expressed as:

$$I_{in}(\mathbf{x}, \lambda) = \int_{\Omega} R_{in}(\mathbf{x}, \lambda) L_{in}(\lambda) S_{in}(\lambda) d\lambda, \tag{3.2}$$

where:

- $R_{in}(\mathbf{x}, \lambda)$ is a scene containing a color checker.
- $L_{in}(\lambda)$ is the illumination properties.
- $S_{in}(\lambda)$ is the device properties of the camera used to capture the scene.
- I_{in} is the resulting image.

To begin the color correction process we choose 24 input points ($[R, G, B]$) from the input image I_{in} and place them into ρ. These input points are chosen to represent one square each of the color checkers 24 squares and will correspond with the ground-truth points ($[X, Y, Z]$) found in $\mathbf{q}$. The points in $\mathbf{q}$ are defined by the manufacturer of the color checker and is therefore always both illumination and device independent. We now want to calculate the

mapping $\mathbf{M}$ that will transform ρ into $\mathbf{q}$. Once we have calculated the mapping $\mathbf{M}$ we can apply it to the input image I_{in} giving us the device and illuminant independent image $I_{independent}$, as shown in Figure 3.1, that can be expressed as:

$$I_{ind}(\mathbf{x}, \lambda) = \int_{\Omega} R_{in}(\mathbf{x}, \lambda) d\lambda. \tag{3.3}$$

This means that if we take another image of the same scene R_{in} under a different illuminant, lets call it L_{new} and with another camera, lets call it S_{new} and we go through the steps described above and demonstrated in Figure 3.1, we should ideally end up with the same final image I_{ind}. This is clearly shown in Equation 3.3 as I_{ind} is only a function of R_{in}. While this method ensures consistent color representation across various devices and illuminants, it does have a significant drawback of relying on an external color checker. The upcoming sections will introduce our novel approach that addresses this limitation.

3.2 Ground-Truth Image Based Color Correction (GTIB-CC)

Building on the discussions in Chapter 2 and the examples provided, it's clear that color correction is fundamentally dependent on ground-truth points. While the most common solution to gathering these ground-truth points is to use an external tool, such as a color checker, we propose a solution that does not rely on such an external tool. The approach we propose involves the use of two distinct images: an *Input Image* and a *Ground-Truth Image*.

The Ground-Truth Image is a snapshot of the scene captured under the desired lighting conditions. The content of this scene isn't as important as its availability; it should be a scene that we can reliably return to during the color correction process. For robotic applications, we recommend using the robot itself as the ground-truth scene. This image provides a reliable source of reference points that are unaffected by changes in illumination or the capturing device.

The Input Image, on the other hand, is the image that we wish to correct. It's captured of the same scene as the Ground-Truth Image, but under different illumination conditions. This is the image we will be manipulating to match the color profile of the Ground-Truth Image.

Our approach, which we've named *Ground-Truth Image Based Color Correction (GTIB-CC)*, allows us to select a number of corresponding points in the two images. These points provide both the ground-truth data and the input data for the color correction algorithm. This method is particularly useful in robotic applications where the robot itself can be used as the ground-truth scene.

The GTIB-CC method offers a unique approach to color correction, providing a flexible and adaptable solution to the challenges posed by varying illumination conditions and capturing devices. The impact of this method on the final image, along with its advantages and limitations, will be demonstrated in the example that follows.

3.2.1 Example of Ground-Truth Image Based Color Correction (GTIB-CC)

This example will illustrate the effect GTIB-CC has on the final representation of the image as well as help highlight some of the advantages, but also limitations, of this approach. This example is also mirrored in Figure 3.2.

In the GTIB-CC method, we begin by capturing a scene under the desired lighting conditions to create the Ground-Truth Image. For this example, we will say that the camera used to capture the scene has the properties S_{gt} and that the illuminant we captured the scene under has the property L_{gt}. Finally, we say that the scene that is captured is called R_{gt}. This gives us the Ground-Truth Image, I_{gt} that can be expressed as:

$$I_{gt}(\mathbf{x}, \lambda) = \int_{\Omega} R_{gt}(\mathbf{x}, \lambda) L_{gt}(\lambda) S_{gt}(\lambda) d\lambda. \tag{3.4}$$

The next step is to capture the Input Image, $I_{in}(\mathbf{x}, \lambda)$, which is the image we wish to correct. The goal is to transform this image to match the Ground-Truth Image, $I_{gt}(\mathbf{x}, \lambda)$. This is achieved by mapping the input illuminant L_{in} and device S_{in} to the reference illuminant L_{gt} and device S_{gt}.

To do this, we select N number of reference points from the images I_{in} and I_{gt}. It is important that these points represent the same scene, as such, we need to avoid any shadows and highlights that appear in one image, but not the other. We also need to avoid any objects that might be unique to one of the images. This is done by creating a map of any differences in the two images, something which will be discussed in Section 4.1.3. If we fulfill this criteria we put the points from I_{in} into ρ and the points from I_{gt} into $\mathbf{q}$. We then use these points to calculate the mapping $\mathbf{M}$ and finally apply this to the Input Image I_{in}.

The mapping $\mathbf{M}$ maps the input illuminant L_{in} and the input device S_{in} to the ground-truth illuminant L_{gt} and the ground-truth device S_{gt}. This means that the Input Image can be expressed as:

$$I_{in}(\mathbf{x}, \lambda) = \mathbf{M} \int_{\Omega} R_{gt}(\mathbf{x}, \lambda) L_{in}(\lambda) S_{in}(\lambda) d\lambda = I_{gt}(\mathbf{x}, \lambda) \tag{3.5}$$

This method allows us to consistently achieve the ground-truth conditions by mapping the current illuminant and device to those of the Ground-Truth Image, provided we have access to the scene R_{gt}. While this does not give us a device and illuminant independent image, this does give us an image with a consistent color representation, which is what we want for computer vision applications.

In the following section we will discuss the proposed workflow of both of these color correction solutions and how they can be used in the workflow of a robotic vision application. We will also describe how we plan on utilizing the power of color constancy as a complement to these methods.

3.3 Proposed Workflow

In the context of an industrial robot, we can integrate two methods into its workflow: the Ground-Truth Image Based Color Correction (GTIB-CC) and the Color Checker Based Color

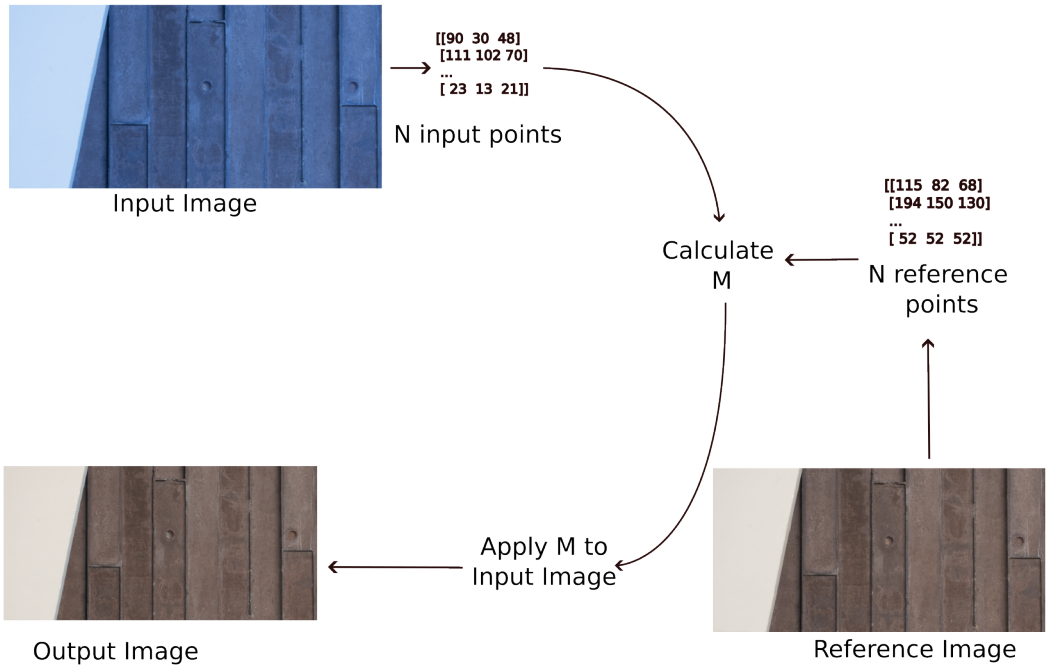


Figure 3.2: Color Correction flowchart. Image adapted from Figure 3, Lai et al[26].

Correction (CCB-CC). The specific workflow varies depending on the method applied. We'll first outline the workflow for CCB-CC, followed by GTIB-CC.

To aid in this description we will consider an industrial robot with a camera mounted to it. There are three "scenes" available to the robot. The first scene is a table with colored blocks placed on it, these blocks will be manipulated by the robot. This is the "working scene". The second scene is the "ground-truth scene", this is a scene showing the robot base. The third and final scene is a scene showing the color checker, which is placed near the "working scene". In this example we assume the robot is tasked with pick-and-place. The setup required for this is a tuning of an object detection algorithm under the current illumination conditions.

With this scenario in mind we will now present the two workflows of this robot, starting with the workflow of CCB-CC.

3.3.1 Workflow of CCB-CC

The CCB-CC workflow initiates with a setup step, that starts by directing the camera towards the "color checker scene". The camera captures an image, locates the color checker within it, and extracts the input points. These points are used, together with the ground-truth points provided by the manufacturer, to calculate a mapping, which is then saved and applied to the images of the "working scene". Optionally, color constancy can be applied at this stage. The final step involves calibrating the object detection algorithm with the color-corrected image. At this point the setup stage is finished.

During operation, the camera periodically returns to the "color checker scene" either at a set interval or when lighting conditions change. It locates the color checker, calculates a new mapping, and applies this mapping to the images of the working scene. Optionally, color

constancy can be applied at this point as well.

3.3.2 Workflow of GTIB-CC

The GTIB-CC workflow starts again with a setup. The setup begins by capturing a ground-truth image of the "ground-truth scene" under the desired illumination conditions. A subsequent image of the "ground-truth scene" under the current conditions is taken to serve as the input image. A mapping is calculated from these two images and applied to the "working scene" images. Optionally, color constancy can be applied at this stage. The object detection algorithm is then tuned using the color-corrected image and at this point the setup is complete.

During operation, the camera periodically captures a new image of the "ground-truth scene" either at a set interval or when lighting conditions change. A new mapping is calculated using the Ground-truth image and the new image, and this mapping is applied to the working scene. Optionally, color constancy is applied at this point.

3.4 Methods for Evaluation of Algorithms Performance

To evaluate the performance of the algorithms, we need a method to measure the proximity of the colors in the final image to a series of ground-truth colors. We decided to define our own method, where we will be calculating the *Mean Square Error (MSE)* between an input array of color values and a ground-truth array of color values, providing a measure of how closely the colors of the input array align with those of the ground-truth array.

However, it's important to note that this method is influenced by the brightness of the image. For instance, if the input array values are derived from an image that is darker than the ground-truth array, the MSE will increase. Given that some color correction and color constancy algorithms can affect the brightness of the image, we will instead aim to measure the MSE independently of brightness.

To understand how we can calculate the MSE independently of brightness we will first introduce how we calculate MSE. We will then introduce a modified version that we call rg-chromaticity MSE. rg-chromaticity MSE is a luminance-independent version of the MSE calculation, providing us with a method for evaluating the proximity of colors without the influence of the brightness of the image.

The details of these methods are presented in the sections below.

3.4.1 Understanding the Calculation of RGB MSE

The Mean Square Error (MSE) is a measure calculated from the difference between two arrays of color values: the input color array and the ground-truth color array. The input color array consists of color values from the image that is either to be color corrected or has already been color corrected, depending on the stage of evaluation.

On the other hand, the ground-truth color values are derived from one of two sources. They can either be the color checker values provided by the manufacturer of the color checker

or extracted directly from the ground-truth image.The next chapter will provide a more detailed explanation of how we collect both the ground-truth and input values. Once we have these two arrays of color values, we can calculate the MSE. This calculation is demonstrated in Algorithm 1:

Algorithm 1 Calculation of RGB MSE

$\mathbf{V_{in}} = [R_{in}, G_{in}, B_{in}] \in \{0, 255\}^3$
$\mathbf{V_{gt}} = [R_{gt}, G_{gt}, B_{gt}] \in \{0, 255\}^3$
$N = 1$
$arrayLength \leftarrow sizeOf(\mathbf{V_{in}})$
while $N \leq arrayLength$ **do**
    $MSE_N = (R_{in,N} - R_{gt,N})^2 + (G_{in,N} - G_{gt,N})^2 + (B_{in,N} - B_{gt})^2$
end while
$MSE = \sum_{n=1}^{arrayLength} MSE_N$

3.4.2 Understanding the Calculation of rg-chromaticity MSE

This algorithm provides a luminance-independent version of the MSE calculation. The key difference lies in the space where we calculate the MSE value: instead of the RGB space, we use the rg-chromaticity space. This adjustment ensures that any differences in luminance are not factored into the calculation, resulting in an MSE value that exclusively depends on the differences in color. The calculation of this algorithm's MSE value is demonstrated in Algorithm 2. Although we have included the calculation for the b-chromaticity value for completeness, it does not need to be explicitly calculated as the total sum $r + g + b$ always equals one:

Algorithm 2 Calculation of rg-chromaticity MSE

$[R_{in}, G_{in}, B_{in}] \in \{0, 255\}^3$
$[R_{gt}, G_{gt}, B_{gt}] \in \{0, 255\}^3$
$N = 1$
$M = 1$
$arrayLength \leftarrow sizeOf(\mathbf{V_{in}})$
while $N \leq arrayLength$ **do**
    $[r_{in,N}, g_{in,N}, b_{in,N}] = [\frac{R_{in,N}}{R_{in,N}+G_{in,N}+B_{in,N}}, \frac{G_{in,N}}{R_{in,N}+G_{in,N}+B_{in,N}}, \frac{B_{in,N}}{R_{in,N}+G_{in,N}+B_{in,N}}]$
    $[r_{gt,N}, g_{gt,N}, b_{gt,N}] = [\frac{R_{gt,N}}{R_{gt,N}+G_{gt,N}+B_{gt,N}}, \frac{G_{gt,N}}{R_{gt,N}+G_{gt,N}+B_{gt,N}}, \frac{B_{gt,N}}{R_{gt,N}+G_{gt,N}+B_{gt,N}}]$
end while
while $M \leq arrayLength$ **do**
    $MSE_{rg,N} = (r_{in,N} - r_{gt,N})^2 + (g_{in,N} - g_{gt,N})^2 + (b_{in,N} - b_{gt,N})^2$
end while
$MSE = \sum_{n=1}^{arrayLength} MSE_{rg,N}$

In the following section we will be giving an introduction to the dataset used during evaluation of the algorithms being investigated.

3.4.3 Dataset Employed for Evaluation

For the testing and evaluation of the color correction and color constancy algorithms, we utilized *The Rendered WB dataset (Set 1)*, a dataset curated by Afifi et al[1]. This comprehensive dataset comprises a vast array of images captured both indoors and outdoors. Each input image in this dataset is accompanied by a calibrated ground-truth image. The dataset includes images both with and without color checkers, making it an ideal choice for this project as we aim to investigate methods that both utilize and do not utilize color checkers. While we considered generating our own dataset as this would have made testing and evaluation more consistent and reliable, the pre-existing *The Rendered WB dataset (Set 1)* offered a depth of variety that would have been time-consuming and challenging to replicate on our own.

Chapter 4
Implementation

This chapter provides a comprehensive guide to the methods used to improve color perception in robotics. It begins with a detailed explanation of the color correction implementation, including automatic color checker detection and point selection for ground-truth image-based color correction. Next, the chapter delves into the application of color constancy algorithms, explaining how they enhance color correction results. The chapter then explores the combination of color correction and constancy algorithms, discussing the implementation of rg-chromaticity MSE and optimized linear weighting for improved outcomes. Finally, the chapter concludes with a discussion on shape detection and color segmentation techniques, which are used during the evaluation of the color correction and constancy algorithms.

Each section provides a step-by-step guide, making this chapter a potentially valuable resource for those interested in enhancing color perception in robotics.

4.1 Color Correction Implementation

The implementation of color correction in this project utilizes a color science library known as *Colour*[11]. This library offers a wide array of tools for handling colors in Python, including color correction algorithms. As outlined in Section 2.3, our investigation focuses on three distinct color correction algorithms:

- Color Correction using Root-Polynomial Regression
- Color Correction using Polynomial Term Expansion
- Color Correction using the Vandermonde Matrix

The Colour library provides implementations of all three of these algorithms. However, in the Colour library these algorithms are named after the author, as such these are called:

- `Finlayson 2015`

- `Cheung 2004`
- `Vandermonde`

The Colour library's implementation of each algorithm shares a similar interface. Depending on the method chosen, either the number of terms or the degree of polynomial expansion can be specified. All algorithms require three common inputs:

- An image to be corrected: This input represents the image that requires color correction.
- An array of input values: These values are extracted either from the color checker found in the input image, or directly from the input image, depending on the method of color correction being used. These values will be matched to the ground-truth values during the color correction process.
- An array of ground-truth values: These values are either given to us from the manufacturer of the color checker, or directly extracted from the ground-truth image, depending on the method of color correction being used. This array serves as the ground-truth against which the input values will be compared and adjusted.

The algorithm `Cheung 2004` also requires the number of terms of the expanded polynomial as an input, these terms must be $\in 3, 4, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22, 35$. The algorithms `Finlayson 2015` and `Vandermonde` both take the degree of the expanded polynomial as an input, which must be $\in 1, 2, 3, 4$. In the following section the details of our implementation is presented.

4.1.1 Implementation Details of Color Correction

The color correction calculation is encapsulated within a function named `calculate_color_correction_matrix`, which computes the color correction matrix for a given image. This function accepts three arguments: `input_values_RGB`, `ground_truth_values_RGB` and `color_checker`. See Figure 4.1 for an illustration of the program flow described below.

The `input_values_RGB` argument is an array of input values in the RGB color space, representing the colors in the image that are to be corrected. The `ground_truth_values_RGB` argument is an array of ground-truth values, also in the RGB color space, which serve as the ground-truth to which the input values should be adjusted. The `color_checker` argument is a boolean indicating whether a color checker is utilized in the color correction process.

The function first checks if a color checker is used. If it is, the function calculates the color correction matrix using the `Cheung 2004` method with root-polynomial regression of terms 7 and 8, and the `Vandermonde` method with a polynomial degree expansion of 2. The resulting matrices are then combined into a single matrix using predefined weights. The combined matrix is returned by the function.

If a color checker is not used, the function calculates the color correction matrix using the `Cheung 2004` method with root-polynomial regression of terms 5 and 7, and the `Vandermonde` method with a polynomial degree expansion degree 2. The resulting matrices

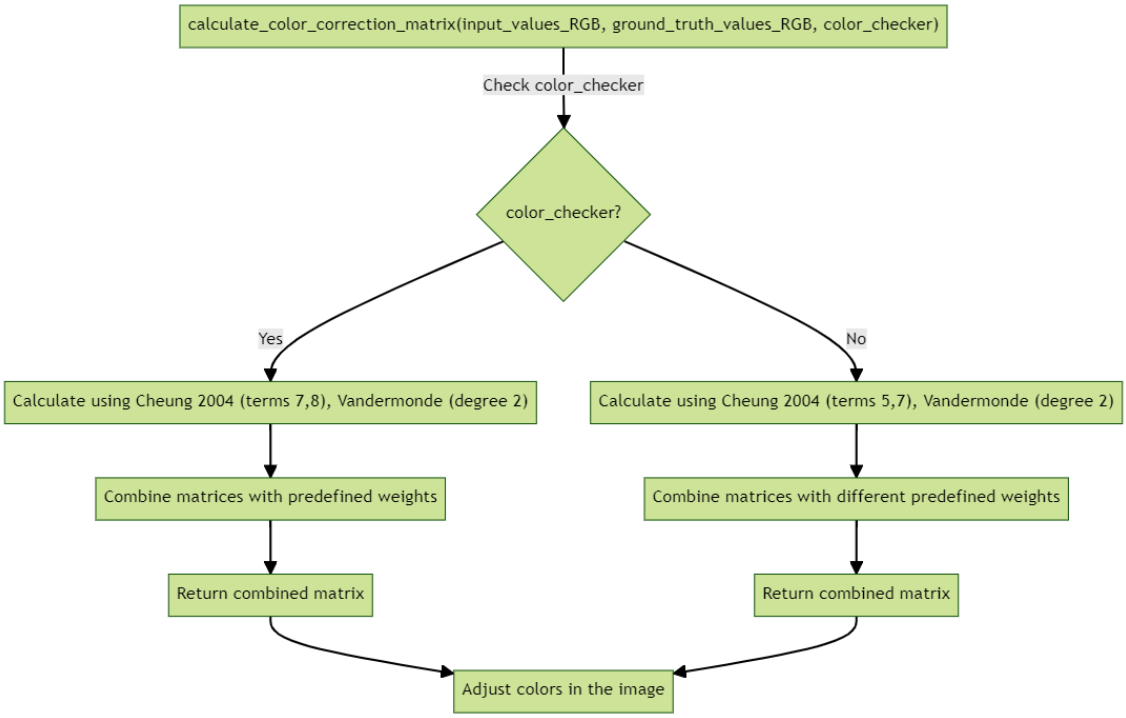


Figure 4.1: Flow Chart of Color Correction Implementation

are combined into a single matrix using a different set of predefined weights. The combined matrix is again returned by the function.

The color correction matrix calculated by this function can be used to adjust the colors in the input image so that they match the ground-truth values as closely as possible. This is typically done by multiplying the color values in the image by the color correction matrix.

For more detailed information regarding the implementation and usage of these algorithms, we refer the reader to the official documentation of the Colour library[12]. We will discuss the performance of these algorithms in Section 6.1. During the performance evaluation, we evaluate each possible term or degree expansion allowed by the algorithm's implementation to find the best performing.

4.1.2 Automatic Color Checker Detection

The automatic color checker detection process utilizes the Macbeth Chart module from the OpenCV library[30]. The key function in this process, `find_color_checker`, takes an RGB input image as an argument. This function creates a color checker detector object and initiates the detection process on the input image, a flow chart of this can be seen in Figure 4.2.

If the color checkers is successfully detected in the image, the function extracts its color values. However, the output from the color checker detector object is not in the desired format for further processing. To address this, the output is passed to another function,

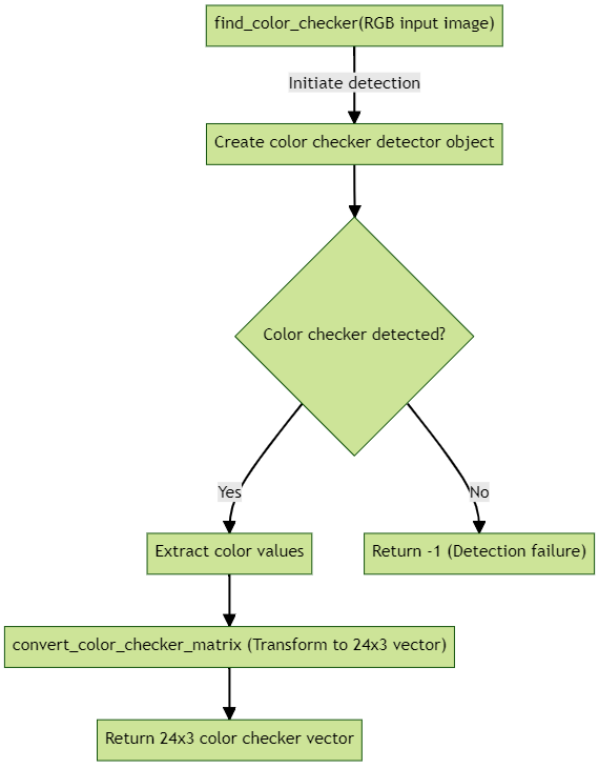


Figure 4.2: Flow Chart of Color Correction Implementation

`convert_color_checker_matrix`.

The `convert_color_checker_matrix` function transforms the color checker values into a 24×3 vector. Each row in this vector corresponds to the RGB value of a specific square on the color checker. The vector's height is 24, reflecting the 24 distinct squares present on the color checker. The values are consistently stored in a left-to-right, top-to-bottom order. This consistent ordering facilitates the straightforward matching of detected values to the corresponding ground truth values provided by the color checker's manufacturer.

These processed values serve as input for both the color correction algorithms and the Mean Squared Error (MSE) calculation. If the color checker is not detected in the image, the `find_color_checker` function returns -1, signaling the detection failure.

Next we will be presenting how we can find ground-truth points from our ground-truth image when utilizing GTIB-CC.

4.1.3 Automatic Point Selection for GTIB-CC

For GTIB-CC we need a systematic way for selecting good ground-truth points. We will begin with making the assumption that the input and ground-truth images are aligned. We do, however make no assumption about the content of each image more than that there are common parts. We do allow, for example, shadows, highlights and new objects to enter the scene. However, we assume that there are some common parts that can still be seen as this is crucial for being able to select reference points. The core idea is that we want to only select reference points at the pixel coordinates that represent objects that are common to both images. An example of how the ground-truth and input images can look is shown in

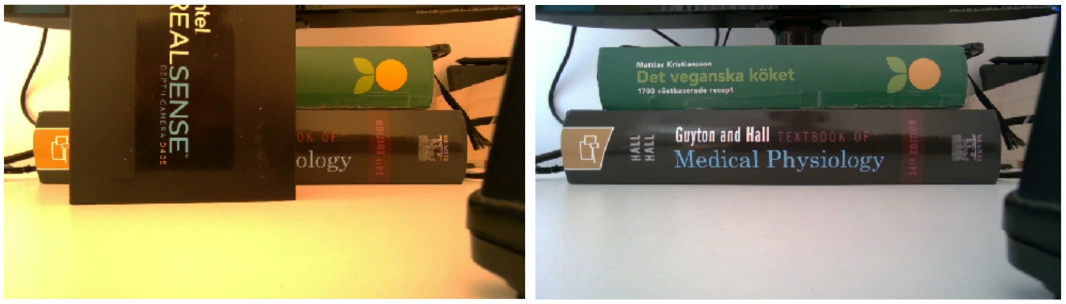


Figure 4.3: Input and Reference Image Example

Figure 4.3.

Finding Common Points in Images

In the process of identifying common points in the images, the initial step involves analyzing the differences between the images. This is achieved by converting the images to grayscale, a step that is crucial to ensure that color variations do not influence the determination of similarities. We defined a function called `create_similarity_binary_map` that performs this task by converting both the input and ground-truth images to grayscale using the OpenCV function `cv.cvtColor`.

Next, the function computes the absolute difference between the grayscale versions of the input and ground-truth images, resulting in a new image (`diff_image`) where each pixel value represents the absolute difference between the corresponding pixels in the two original images. The pixel values in this difference image range from $[0, 255]$, with larger values indicating greater differences and smaller values indicating smaller differences.

To isolate the common points in the images, a threshold is applied to the difference image, generating a binary image. This is achieved using the OpenCV function `cv.threshold`, which applies an inverse binary threshold to `diff_image`. The thresholding value we found showed the best result was 7, meaning that all pixel values in `diff_image` that are less than or equal to 7 are set to 1 in the binary image, and all other pixel values are set to 0.

In this binary image, a pixel value of 1 indicates a potential reference point, while a pixel value of 0 indicates a point that cannot be used as a reference. By adjusting the thresholding value, we can control the degree of similarity required for a pixel to be considered a potential reference point.

Figure 4.4 provides an example of such a binary image, which was generated using the ground-truth and input images shown in Figure 4.3 and a thresholding value of 7. In this binary image, black pixels represent non-reference points, while white pixels represent potential reference points.

Selecting Reference Points

The selection of reference points aims to ensure a comprehensive representation of the image's content while avoiding any local peculiarities. This is achieved by choosing points that

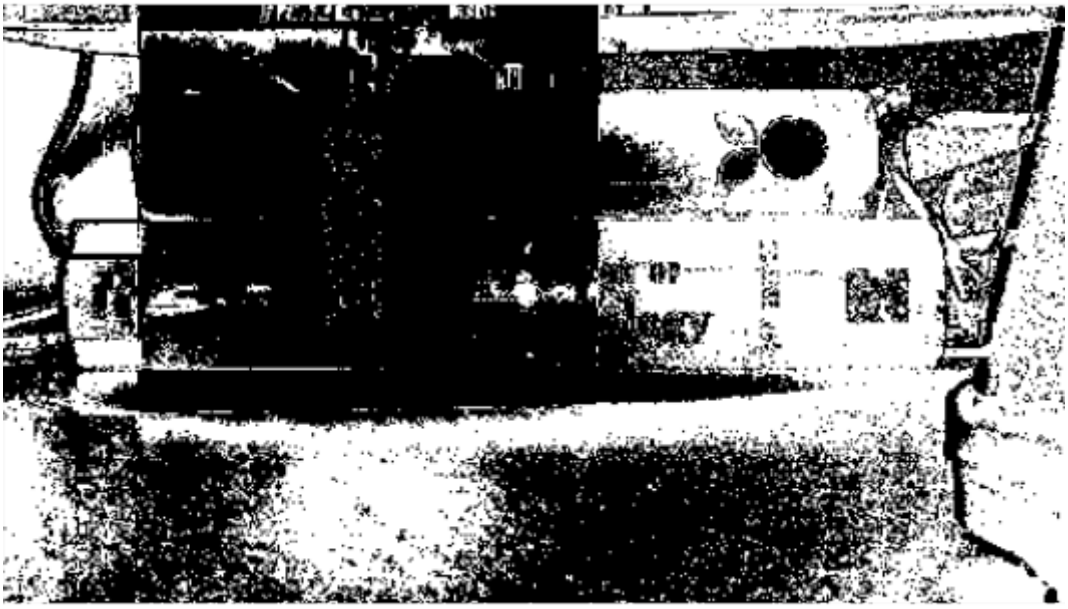


Figure 4.4: Binary Image Example. Created by taking the difference between the input and reference image in Figure 4.3.

are evenly distributed across the image and only where the binary map has a value of one, indicating potential reference points.

The function `pick_ref_points` is used to select these reference points. It first calculates the number of points to select along each dimension of the image and the spacing between these points. It then iterates over the points to select along each dimension of the image, computes the coordinates of each point, and checks if each point is a valid reference point by calling the `is_valid_ref_point` function.

The `is_valid_ref_point` function checks if a given reference point is valid by verifying the condition that the binary map has a value of one at the point's location, which indicates that the point is within the image bounds. If it passes this condition, it is considered valid.

If a point is deemed valid, it is added to the list of selected reference points. Once all points have been evaluated, the function returns the list of selected reference points. These points are then used to extract the corresponding pixel values from both the input image and the ground-truth image. These values are stored in two $N \times 3$ vectors, where N is the number of points, and are used in the same way as the color checker values when performing color correction and Mean Square Error (MSE) calculations.

Figure 4.5 illustrates the selected reference points overlaid on the input image from Figure 4.3, using the binary image from Figure 4.4 as a guide. This visual representation aids in understanding how the selected reference points are distributed across the image.

4.2 Implementation of Color Constancy

The implementation of color constancy in this project leverages the additional photo processing algorithms provided by the OpenCV library[29]. This library offers three classes for

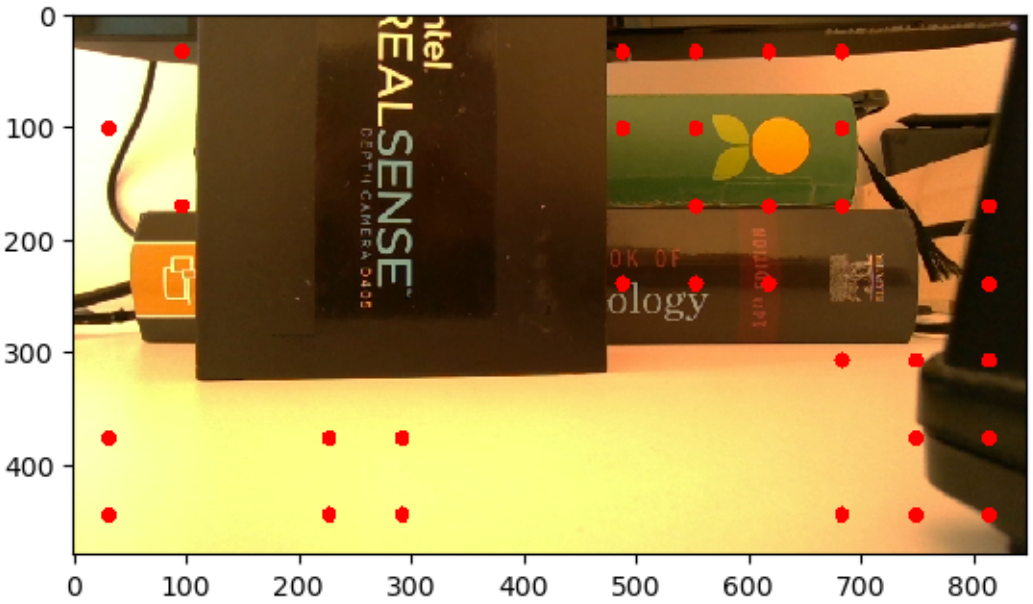


Figure 4.5: Input Image with Reference Points Drawn

color constancy: `GrayworldWB`, `LearningBasedWB`, and `SimpleWB`. All these classes inherit from the base class `WhiteBalancer` (white balance correction is an alternative term for color constancy).

The function `apply_color_constancy` is defined to apply these color constancy methods to an RGB input image. The function applies the three color constancy methods to the input image. This is achieved by creating an instance of each color constancy class (`greyworldWB`, `simpleWB`, and `learningBasedWB`) and invoking the `balanceWhite` method on each instance with the image as input. This method applies the corresponding color constancy algorithm to the image and returns three corrected images, one for each method.

The function then combines the three corrected images into a single image. This is accomplished by taking a weighted sum of the three images, the calculation of these weights is discussed in Section 4.3.

This comprehensive approach ensures that the final image benefits from the strengths of each white balance correction method, leading to a more accurate and robust color constancy.

In the next section we will discuss how we can combine color correction algorithms and how we can combine color constancy algorithms. We will also be presenting how we implemented the rg-chromaticity MSE calculation.

4.3 Combining Multiple Color Correction and Constancy Algorithms

As previously discussed, the combination of several color constancy algorithms can yield superior results compared to each algorithm operating independently. We also want to investigate if this holds true for color correction. In this section, we present our approach to combining algorithms, starting with the implementation details of calculating the rg-chromaticity

MSE values, as this is a crucial component in determining the ideal combination of algorithms.

4.3.1 Implementation of rg-chromaticity MSE

The function designed to calculate the rg-chromaticity Mean Squared Error (MSE) is called `rg_chromaticity_MSE` and accepts two sets of RGB values as input: `input_values` and `ground_truth_values`. Initially, it scales the RGB values of both the input and ground-truth sets to be in the rg-chromaticity space. This is achieved by dividing each RGB component by the sum of all RGB components for each color, see Algorithm 2. This process is applied to each color in the input and ground-truth sets, resulting in two new sets of scaled values in rg-chromaticity space. The function then calculates the MSE between these two sets of rg-chromaticity space values using the `calc_MSE` function.

The `calc_MSE` function calculates the Mean Squared Error between two sets of values: `input_values` and `ground_truth_values`. It accomplishes this by subtracting the ground-truth values from the input values, squaring the result, and then taking the mean of these squared differences, the function then returns this mean squared error.

In summary, the `rg_chromaticity_MSE` function calculates the luminace invariant Mean Squared Error between two sets of RGB values, which is useful for comparing the similarity of colors in different images. The `calc_MSE` function is a helper function used to calculate the Mean Squared Error.

We now move on to the implementation of the function used to calculate the optimal weighting for combining algorithms.

4.3.2 Optimized Linear Weighting

Our goal is to optimally combine the results of the three different image enhancement methods discussed above - namely, CCB-CC, GTIB-CC, and Color Constancy (CC). We calculate three different weightings, one for each method. Each method takes an original image as input and transforms this image, producing three different resultant images for every single input image, which we refer to as `image_1`, `image_2`, and `image_3`. To effectively merge these three images, we create a final composite image for each input image through a linear combination of the three resultant images from the algorithms.

The main challenge here lies in determining the most suitable weighting for each algorithm in the linear combination, denoted as $[w_1, w_2, w_3]$. We aim to select these weights in such a way that the final composite image is as close as possible to the ground-truth image, with the 'closeness' quantified through the rg-chromaticity Mean Squared Error (MSE). The lower the MSE, the closer the composite image is to the ground-truth image.

Two conditions must be satisfied by the chosen weights:

- The sum of the weights $w_1 + w_2 + w_3 = 1$, which ensures that the total intensity of the final image remains the same.
- The individual weights are all non-negative: $w_1 \geq 0, w_2 \geq 0, w_3 \geq 0$.

To identify the best weights, we first initialize them to random values that satisfy these conditions. Then we employ a gradient descent process, iterating through potential sets of

weights and calculating the MSE of the composite images formed by using these weights. Specifically, we explore the neighboring weights and select the ones that lead to the smallest MSE. We continue this process until we find a set of weights where no neighbors provide an improved MSE.

This weight determination process is performed on a large dataset, and the effectiveness of the chosen weights is then validated during testing. The output of the code is the MSE value of the optimal weighting combination and the respective weightings for each of the three images. In doing so, the algorithm enables us to seamlessly combine different image enhancement methods into a single, optimized process.

We conclude this chapter with a brief overview of the shape detection used during the real-world scenario evaluation portion.

4.4 Implementation of Shape Detection and Color Segmentation

The primary objective of this application is to assess the effectiveness of color correction and color constancy algorithms in real-world situations. As such, while we recognize that the performance of an HSV-based (HSV stands for hue, saturation, value and is an alternative method of representing a color image) segmentation approach tends to be suboptimal under varying illumination conditions our goal is to demonstrate that by employing both color correction and color constancy techniques, we can mitigate this sensitivity to changes in lighting. In doing so, we aim to demonstrate the overall increase in performance.

The process begins by receiving an image which is then converted to HSV (Hue, Saturation, Value) color space. HSV color space is frequently used in computer vision applications as it separates image intensity from color information, making it more resilient to lighting variations.

Subsequently, the code defines the hue, saturation, and value ranges. These ranges are utilized to create masks for different objects in the image. Each object has a specific HSV value, which is employed to create a binary mask where the pixels of the object fall within the specified HSV range.

The code then specifies the HSV values for various objects (e.g., banana, cups, post-its, pen, apple, red ring), see Figure 6.4 for an example of the scene being investigated. For each object, it calculates the lower and upper HSV values that define the range for that object. For the red ring and apple, two ranges are defined because their hue values are close to either 0 or 180, and as such the hue value wraps around in the HSV color space.

The `cv2.inRange` function is used to create a binary mask for each object. The function checks if the HSV value of each pixel falls within the specified range. If it does, the pixel is set to 255 (white), otherwise, it is set to 0 (black).

All the masks are then combined to create a final mask that includes all the objects. This mask is then processed using morphological operations (closing and opening) to remove noise. The kernel size for these operations is defined as an 11x11 matrix of ones.

The `cv2.bitwise_and` function is used to apply the mask to each pixel in the original RGB image, resulting in an image where only the objects of interest are visible, and everything else is black.

Finally, the code finds contours in the mask using the `cv2.findContours` function. For each contour, it calculates the center point and draws the contour and the center point on the original RGB image, it also adds a text label for each shape. The final image is then displayed on screen.

Chapter 5
Evaluation

This chapter outlines the evaluation methods employed to assess the performance of the color correction and color constancy algorithms discussed in the previous sections. The results of these evaluations will be presented in the subsequent chapter (see Chapter 6).

Our evaluation strategy comprises two distinct approaches. The first is a statistical approach, where we utilize a large dataset to measure the performance of the various color correction and color constancy algorithms. The second is an experimental approach, where we assess the performance of the methods in a real-world scenario.

5.1 Evaluation Using Dataset

The evaluation process utilizes the *The Rendered WB dataset (Set 1)*[1], as detailed in Section 3.4.3. We will be using 300 images from this dataset during out evaluation. We assess three distinct methods:

- Color Checker based Color Correction (CCB-CC).
- Ground-Truth Image based Color Correction (GTIB-CC).
- Color Constancy (CC).

Each of these methods comprises several image correction algorithms that require evaluation, as outlined in Sections 4.1 and 4.2. The evaluation process is as follows:

1. For each image in the dataset, we identify and measure N distinct reference points. Depending on the method under evaluation, these points will either be the 24 points found in the color checker or N points directly from the input image, as described in Section 4.1.3. If we are using the latter method, we also select the same N points in the ground-truth image.

2. We then calculate the initial *rg-chromaticity MSE* value using these points. This metric quantifies the color difference between the input colors and the ground-truth colors.

3. Next, we apply each of the color correction algorithms (outlined in Section 4.1) or the color constancy algorithms (outlined in Section 4.2) to each image individually.

4. From the output of each color correction method, we calculate a new *rg-chromaticity MSE* value. This value represents the color difference between the corrected colors and the ground-truth colors.

5. After performing this process for all 300 images from the dataset, we calculate the average *rg-chromaticity MSE* for each color correction method, as well as for the input image itself.

6. To facilitate comparison, we normalize these average *rg-chromaticity MSE* values such that the input image's value is equal to 1. This normalization allows us to assess the relative improvement or degradation in color accuracy achieved by each color correction method.

5.1.1 Evaluation of Combined Methods

To assess the effectiveness of combining methods, we selected the top three techniques from each of the methods evaluated previously. The combination process follows the approach outlined in Section 4.3. Subsequently, we calculated the average *rg-chromaticity MSE* across the 300 tested images. Finally, we normalized this value to ensure that the *rg-chromaticity MSE* of the input image equaled 1. This normalization process facilitates a relative comparison of the performance improvement or degradation achieved by the combined methods.

5.2 Experimental Setup

For our experimental testing, we attached a camera to an ABB GoFa robotic arm[20]. We programmed the robotic arm to alternate between two positions. The first position, referred to as the "working position," displayed a scene with multiple colored objects and a color checker. The second position, known as the "ground-truth position," was chosen to capture images of the robot base, which are necessary for Ground-Truth Image based Color Correction (GTIB-CC).

We collected images under various scenarios and lighting conditions. The scenarios included a dark room with a light placed above the scene (Scenario 1), a dark room with a light placed at the robot base (Scenario 2), and a bright room with a light placed at the robot base (Scenario 3). For each scenario, we used eight different light colors:

- White ("Neutral")
- Blue
- Green
- Purple

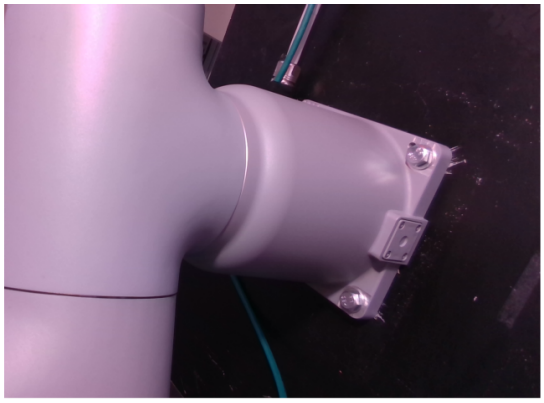
- Red
- Teal
- Orange
- Yellow

For each combination of scenario and light color, we captured an image in both the "working position" and the "ground-truth position". Some examples of the captured images can be seen in Figure 5.1.

After capturing the necessary images, we proceeded with their evaluation. As this test is designed to simulate a real-world scenario, we assessed the relative improvement of an object detection algorithm with and without image enhancement. For each testing scenario, we calibrated the object detection algorithm using the white light, which is intended to replicate a neutral lighting situation.

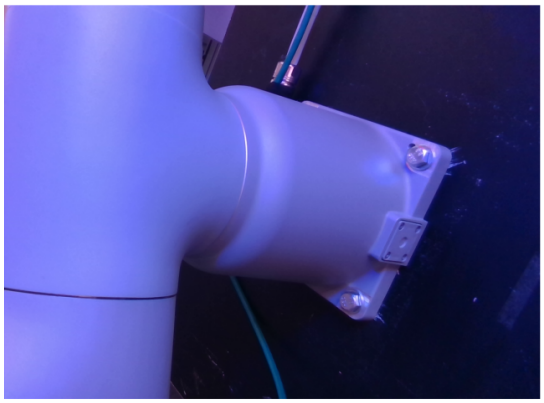
The calibration process begins by applying the appropriate calibration. For instance, if we are evaluating Color Checker based Color Correction (CCB-CC), we first detect the color checker in the "working position" image under neutral lighting conditions, apply color correction, and then tune the object detection algorithm. We then switch to the image taken under blue light, turn off CCB-CC, run the object detection algorithm, and count the number of objects detected. Next we once again run CCB-CC, detecting the color checker, this time under blue light, and apply the color correction. We then run the object detection algorithm and count the number of objects detected after calibration. We repeat this process for every scenario, every color and every method.

We also tested whether combining color correction with color constancy could improve performance. To do this, we followed the same calibration and testing steps as detailed above. However, we also applied color constancy before tuning the object detection algorithm.



Ground-Truth Image (Neutral)

Working Scene Image (Neutral)



Input Image (Blue)

Working Scene Image (Blue)

Figure 5.1: Example of Images Captured During Experimental Data Collection

Chapter 6
Result

In this chapter we will present the result of both the evaluation done on the dataset and the result of the real world experiments.

This chapter presents the findings from the evaluation of various color correction and color constancy algorithms in the context of robotics. It begins with the results from the dataset evaluation, where the performance of different methods such as Color Checker based Color Correction (CCB-CC), Ground-Truth Image Based Color Correction (GTIB-CC), and Color Constancy (CC) are discussed.

We then transition into the results from real-world tests, providing insights into the practical application of these methods. This includes the results of using only color correction in the setup, as well as the results when combining color correction with color constancy.

We conclude with a discussion section, where the performance of color correction and color constancy algorithms, real-world testing results, and the impact of both color correction and color constancy in the setup are analyzed.

6.1 Dataset Evaluation Results

This section details the findings from our evaluation of the dataset using three different methods: Color Checker based Color Correction (CCB-CC), Ground-Truth Image Based Color Correction (GTIB-CC), and Color Constancy (CC).

6.1.1 Color Checker based Color Correction (CCB-CC)

Figure 6.1 illustrates the evaluation results of the CCB-CC method. It particularly highlights the top three high-performing algorithms and their combined result. As shown in the graph, the superior-performing algorithms are `Cheung 2005` with 7 and 11 root-polynomial term

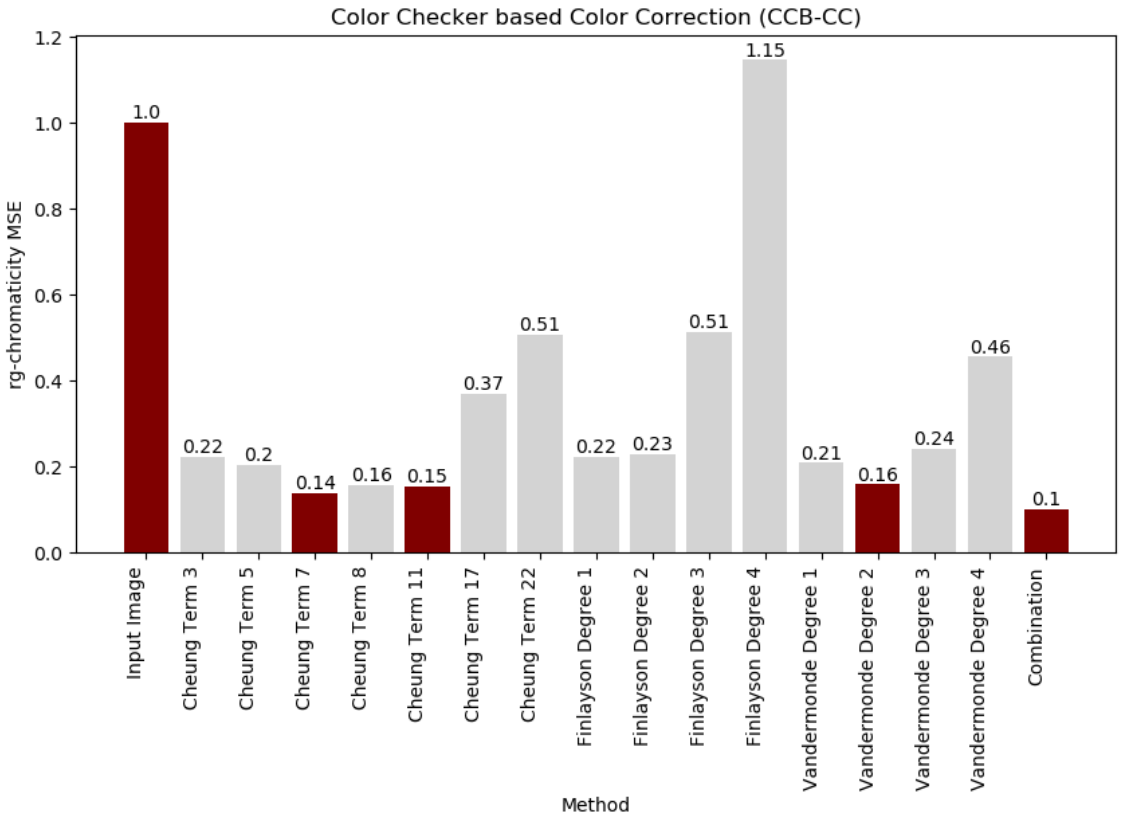


Figure 6.1: Normalized rg-chromaticity MSE for Color Checker based Color Correction (CCB-CC)

expansions and `Vandermonde` with 2 polynomial degree expansions. Notably, the graph also demonstrates that some algorithms exhibit a decline in performance as the number of polynomial or root-polynomial expansions increase. This trend suggests an increased sensitivity to noise and other measurement errors in these algorithms.

6.1.2 Ground-Truth Image Based Color Correction (GTIB-CC)

As depicted in Figure 6.2, the GTIB-CC method's evaluation presents the best three algorithms and their combined result. The most efficient algorithms in this case are `Cheung 2005` with 5 and 7 root-polynomial term expansions and `Vandermonde` with 2 polynomial degree expansions, the degradation effect is more pronounced here, especially at high term or degree expansions. We attribute this to the increased noise and other measurement errors inherent in this method. Furthermore, the overall performance improvement with this method is lower, which we believe is due to the higher measurement error introduced.

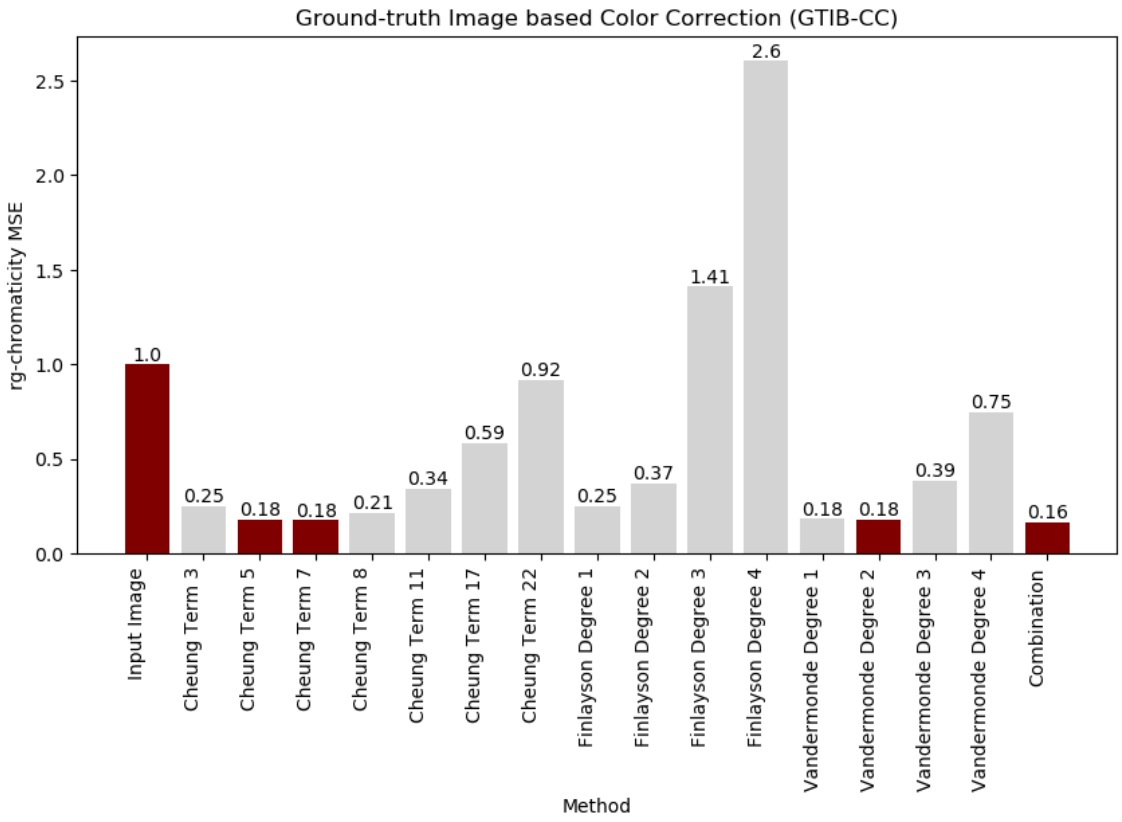


Figure 6.2: Normalized rg-chromaticity MSE for Ground-Truth Image Based Color Correction (GTIB-CC)

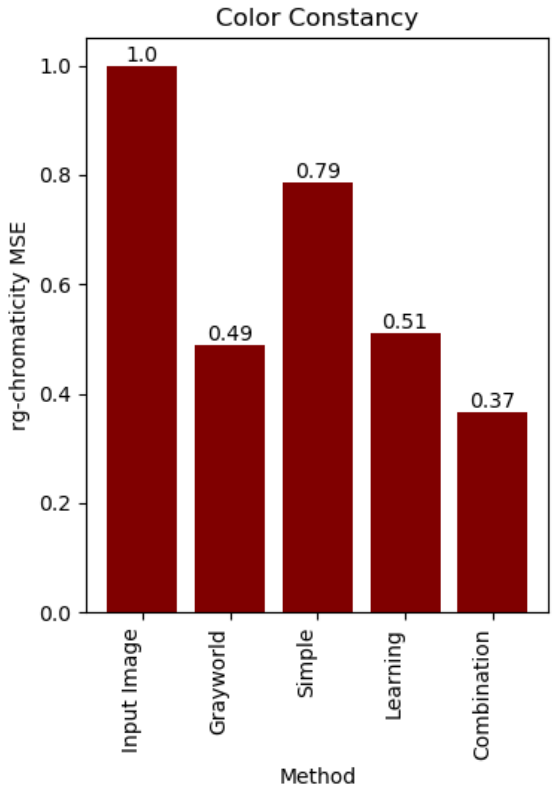


Figure 6.3: Normalized rg-chromaticity MSE for Color Constancy (CC)

6.1.3 Color Constancy (CC)

Figure 6.3 portrays the results of the CC method evaluation. The data clearly indicates a significant advantage when the algorithms are combined.

6.2 Real-World Test Result

This section documents the real-world scenario test results. Our experimental setup has four distinct conditions, differentiated by the use of different color correction methods and the application of color constancy. In Figure 6.4, the setup of the GTIB-CC and GTIB-CC-CC method can be seen and in Figure 6.5 the setup of the CCB-CC and CCB-CC-CC methods can be seen. In Figure 6.6 and 6.7, the effect of applying each respective method to an image under a blue illuminant can be seen.

In the tables provided, we display the outcomes of object detection using various methods, with the total number of objects that can be detected capped at 13. Each table includes the following results:

- **No Correction**: This represents the outcome when no adjustments are made to the image before executing the object detection algorithm.
- **Color Correction (GTIB-CC or CCB-CC)**: This shows the result of applying color

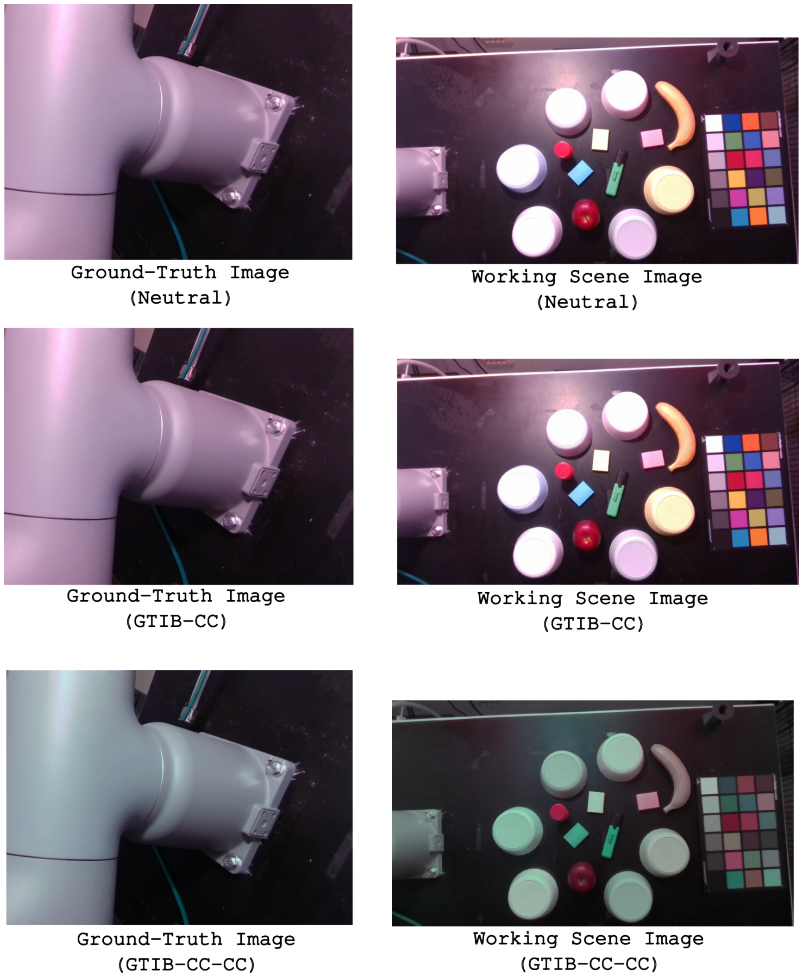


Figure 6.4: GTIB-CC and GTIB-CC-CC Setup

correction to the image before running the object detection algorithm. The specific method used is indicated by either "GTIB-CC" or "CCB-CC."

- **Color Constancy (CC)**: This entry details the outcome of applying only color constancy to the image prior to running the object detection algorithm.
- **Color Correction and Constancy (GTIB-CC-CC or CCB-CC-CC)**: This illustrates the result of applying both color correction and color constancy to the image before executing the object detection algorithm. The specific method used is denoted by either "GTIB-CC-CC" or "CCB-CC-CC."

6.2.1 Result of Only Using Color Correction In Setup

In Table 6.1, we present the results of solely using GTIB-CC (Ground-Truth Image Based Color Correction) in the setup of our object detection algorithm. As can be observed from the table, the performance fluctuates significantly across different scenarios. We hypothesize this variability may be due to the relative placement of the light source with respect to the scene and the robot as well as noise in the measurement data. It is also worth noting that

Figure 6.5: CCB-CC and CCB-CC-CC Setup

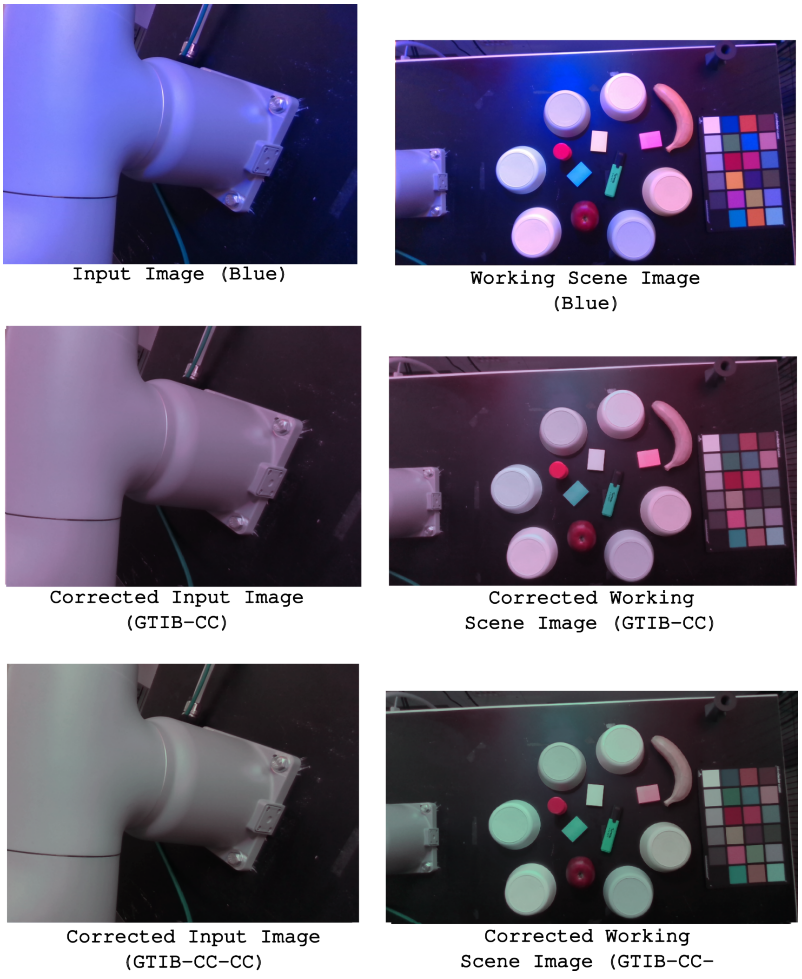


Figure 6.6: Effect of Utilizing GTIB-CC and GTIB-CC-CC

Figure 6.7: Effect of Utilizing CCB-CC and CCB-CC-CC

this setup did not result in any significant increase in performance. A detailed analysis and interpretation of these results will be provided in the following section.

Table 6.1: Number of Detected Objects Using GTIB-CC During Setup

Illuminant	No Correction	GTIB-CC	Color Constancy (CC)	GTIB-CC-CC
Blue	6	8	2	4
Green	4	6	3	4
Purple	8	9	2	4
Red	7	4	4	6
Teal	5	2	2	2
Orange	6	8	6	5
Yellow	8	6	6	5
Scenario (1)	**6.29**	**6.14**	**3.57**	**4.29**
Blue	4	0	3	0
Green	3	1	3	2
Purple	6	3	3	3
Red	5	5	8	4
Teal	5	2	3	0
Orange	10	2	5	4
Yellow	8	4	7	0
Scenario (2)	**5.86**	**2.43**	**4.57**	**1.86**
Blue	3	4	5	3
Green	9	0	9	0
Purple	5	0	5	0
Red	5	1	6	0
Teal	6	1	4	0
Orange	6	0	5	0
Yellow	8	2	6	3
Scenario (3)	**6.0**	**1.14**	**5.71**	**0.86**
Result	**6.05**	**3.24**	**4.62**	**2.34**

The second table, Table 6.2, provides different results. The baseline performance is notably worse than the first table, however, the color corrected version (CCB-CC) shows a considerable improvement. A discussion on the reasons behind these observations will be covered in the next section.

6.2.2 Color Correction and Constancy in Setup

This section presents the test results from the setup that combines both color correction and color constancy methods. This setup involves applying both color correction and color constancy to the image of the scene prior to tuning the object detection algorithm.

In Table 6.3, the results of applying GTIB-CC-CC (Ground-Truth Image Based Color Correction + Color Constancy) are presented. Comparing these results with Table 6.2, we can see a small performance loss when using GTIB-CC alone, which was anticipated. We can also

Table 6.2: Number of Detected Objects Using CCB-CC During Setup

Illuminant	No Correction	CCB-CC	Color Constancy (CC)	CCB-CC-CC
Blue	1	12	2	12
Green	0	6	2	3
Purple	1	11	2	11
Red	0	9	1	9
Teal	0	6	1	8
Orange	0	8	0	3
Yellow	0	8	2	2
Scenario (1)	**0.29**	**8.57**	**1.43**	**6.86**
Blue	1	12	3	11
Green	2	13	4	9
Purple	0	13	5	13
Red	0	11	5	9
Teal	3	12	4	13
Orange	1	11	4	8
Yellow	0	12	3	5
Scenario (2)	**1.0**	**12.0**	**4.0**	**9.71**
Blue	2	7	1	5
Green	1	8	4	6
Purple	2	8	2	8
Red	2	9	3	7
Teal	1	6	3	7
Orange	2	7	0	6
Yellow	1	8	4	7
Scenario (3)	**1.57**	**7.57**	**2.43**	**6.57**
Result	**0.95**	**9.38**	**2.62**	**7.71**

observe a slight performance gain when GTIB-CC-CC was used, however, this improvement is less than we initially anticipated. More surprisingly, there is a significant performance gain when only Color Constancy is applied to the image.

On the other hand, the results in Table 6.4 show a similar pattern, but not the same, to that of GTIB-CC-CC. We observe a performance boost for CCB-CC-CC and a performance loss for CCB-CC. However, it is quite surprising to notice a performance loss when only Color Constancy is applied.

6.2.3 Conclusion of Real-World Test

In conclusion, the application of color correction and color constancy methods can have varied impact on the performance of the object detection algorithm. These findings are subject to several variables and need to be analyzed further. This analysis and a more comprehensive discussion will be provided in the following sections.

Table 6.3: Number of Detected Objects Using GTIB-CC-CC During Setup

Illuminant	No Correction	GTIB-CC	Color Constancy (CC)	GTIB-CC-CC
Blue	4	7	5	6
Green	6	3	8	6
Purple	2	4	7	9
Red	2	3	11	9
Teal	4	0	9	4
Orange	3	6	10	7
Yellow	4	0	12	4
Scenario (1)	**3.57**	**3.29**	**8.86**	**6.43**
Blue	3	0	7	0
Green	3	0	10	0
Purple	2	0	6	4
Red	3	1	7	11
Teal	7	2	8	7
Orange	3	3	10	4
Yellow	4	2	7	3
Scenario (2)	**3.57**	**1.14**	**7.86**	**4.14**
Blue	8	4	10	4
Green	3	0	9	0
Purple	4	0	10	0
Red	4	0	9	0
Teal	4	0	6	0
Orange	4	1	9	1
Yellow	5	2	10	3
Scenario (3)	**4.57**	**1.0**	**9.0**	**1.14**
Result	**3.9**	**1.81**	**8.57**	**3.9**

6.3 Discussion

The performance degradation due to noise, the real-world testing results, and the impact of both color correction and color constancy in setup will be discussed further in this section. We will begin by discussing the degradation due to noise observed in the color correction algorithms.

6.3.1 Performance Degradation in Color Correction Algorithms

Our analysis suggests that the performance degradation seen in both Figure 6.2 and 6.1 is due to the increasing computational complexity and the propagation of approximation errors which is often experienced when executing more polynomial term expansions or root-polynomial expansions [3]. This is analogous to overfitting, where a algorithm is overtrained on the dataset, resulting in a negative impact on the models performance on previously un-

Table 6.4: Number of Detected Objects Using CCB-CC-CC During Setup

Illuminant	No Correction	CCB-CC	Color Constancy (CC)	CCB-CC-CC
Blue	1	7	2	12
Green	0	8	2	7
Purple	0	11	3	12
Red	0	12	1	13
Teal	1	5	3	6
Orange	0	10	1	8
Yellow	2	11	2	13
Scenario (1)	**0.57**	**9.14**	**2.0**	**10.14**
Blue	1	10	3	11
Green	2	11	2	7
Purple	0	12	2	13
Red	0	12	0	10
Teal	2	10	3	12
Orange	0	10	0	6
Yellow	0	11	3	10
Scenario (2)	**0.71**	**10.86**	**1.86**	**9.86**
Blue	1	6	4	9
Green	2	7	3	7
Purple	3	6	3	8
Red	2	7	4	8
Teal	1	7	2	8
Orange	2	6	3	7
Yellow	3	6	3	9
Scenario (3)	**2.0**	**6.57**	**3.14**	**8.0**
Result	**1.09**	**8.86**	**2.33**	**9.33**

seen data. What this means in our case is that as the polynomial terms or the root-polynomial terms are expanded further, the algorithm becomes more and more sensitive to any noise in the measurements taken.

This may also explain why the GTIB-CC method exhibits a more noticeable impact as it exerts less control over the ground-truth data and the input data. On the contrary, the CCB-CC method, which employs a color checker, uses calibrated ground-truth data and a calibrated external tool for data collection. We will discuss a possible solution to this in the next chapter.

6.3.2 Discussion on the Real-World Testing Results

The real-world testing reveals that the performance variability among different contexts and environments should be a significant consideration for anyone looking to use this method. Key factors such as the relative positioning of the light source with respect to the scene and robot have been found to influence the outcomes. This is of extra importance to the GTIB-CC method, for instance, if the light source is closer to the ground-truth scene, than to

the scene being evaluated, this will cause the color correction algorithm to overcompensate, leading to decrease in performance. This also tells us that the ground-truth scene need to be chosen to be as representative to the working scene as possible. This means that the positioning of any light sources needs to be chosen with care to ensure the best possible performance.

We also saw that the GTIB-CC method did not yield any performance advantages over the baseline or the use of CCB-CC, CCB-CC-CC and Color Constancy (CC). We do however believe that with further development this method could yield significantly better performance than shown here as we have identified several areas where the method could be improved, what this development could entail will be discussed in detail in the following chapter.

A clear difference in the baseline performance of GTIB-CC and GTIB-CC-CC from that of CCB-CC and CCB-CC-CC can clearly be seen. This is due to the fact that before calibrating the object detection algorithm color correction is applied to the image. When CCB-CC is applied this causes the image to be heavily altered as the image is now device and illumination independent. When the baseline value is calculated no color correction of color constancy is first applied, meaning that the object detection is first tuned on a heavely alrered image and then ran on a not so heavely altered image. When using GTIB the difference between a non-color corrected image and a color corrected image is not as large and therefore the baseline is not as affected.

The performance disparity in the baseline performance of GTIB-CC and GTIB-CC-CC as compared to CCB-CC and CCB-CC-CC is evident. This stems from the preliminary application of color correction to the image before calibrating the object detection algorithm, where using CCB-CC before the calibration results in a more pronounced alteration as compared to using GTIB-CC before the calibration. This means that the object detection algorithm calibrated using CCB-CC has been calibrated on an image with more pronounced alteration than that calibrated using GTIB-CC. When the baseline value is calculated the object detection application is run on an unaltered image, causing the large performance disparity between GTIB-CC and GTIB-CC-CC as compared to CCB-CC and CCB-CC-CC.

We did however see that CCB-CC and CCB-CC-CC yielded significant performance boosts over the baseline. This is no major surprise, as using a color checker is the most common method for executing color correction. However, it did confirm that combining the result of multiple color correction algorithms has merit.

6.3.3 Impact of Both Color Correction and Color Constancy in Setup

The role of color correction and color constancy in a given setup brings about intriguing results. While combining these two elements sometimes results in a performance increase, the margin of improvement in these cases are not large enough to be deemed significant. This could be due to implementing both simultaneously introducing conflicting signals or adding noise to the system. Interestingly, a more substantial performance gain is witnessed when only color constancy is applied, which might mean that for some applications, only using color constancy might be the most effective method to achieve stable color representation. Unpacking the reasons behind these observed results is essential to further fine-tune

the systems and maximize their performance.

Chapter 7
Conclusions

We will begin this chapter by recapping the research problem and questions. We will then move on to summarizing our findings, relating back to the research problem as we go. Next we will be discussing the implications of our findings in the broader context of the field of color perception in robotics. We will then be giving an open account of any limitations of this study. Based on that, we will be giving suggestions on areas where future research could improve the result achieved, or address some of the limitations. Finally, we will conclude the chapter by summarizing what this research has contributed to the field of robotic color perception.

7.1 Recapping of Research Problem and Questions

The research problem this thesis addresses is the challenge of optimizing color perception in robotics by integrating color correction and constancy. As stated before, color, while a potent feature in computer vision applications, is susceptible to changes in illumination conditions, which can significantly undermine the effectiveness of computer vision algorithms. Therefore, it is crucial to attempt to neutralize the influence of the illuminant, which is addressed by color correction and color constancy.

The primary research questions that guided this study are:

- What is the viability of using a robot base as ground-truth over a color checker board for color correction?
- Can the result of color correction be further improved by the use of color constancy?
- How does the position of the light sources affect the color correction reference?

The investigation focused on enhancing the reliability of color perception in robots that utilize computer vision applications, achieved by integrating color correction techniques with color constancy methods. The research explored the implementation of color correction using external tools, such as a color checker as a reference, and examined a novel approach to color correction that does not rely on external tools but instead utilizes the robot's surrounding environment. A comparative evaluation was conducted to assess the efficiency of using external tools versus relying solely on the environment.

7.2 Summary of Findings

The results of the study, as presented in Chapter 6, were derived from both dataset evaluations and real-world experiments. Three primary methods were tested: Color Checker based Color Correction (CCB-CC), Ground-Truth Image Based Color Correction (GTIB-CC), and color constancy (CC).

The evaluation done with the aid of the dataset helped guide which algorithms where further investigated in the real-world scenario testing. What the dataset evaluation showed was that for CCB-CC, the three best performing algorithms were `Cheung 2005` with 7 and 11 term expansions and `Vandermonde` with 2 polynomial degree expansions. For GTIB-CC the three best performing were `Cheung 2005` with 5 and 7 term expansions and `Vandermonde` with 2 polynomial degree expansions. We also saw for all the methods (CCB-CC, GTIB-CC and CC) that combining algorithms increased the total performance. We decided to linearly combine the three best performing algorithms respectively. These linearly combined algorithms is also what was used during the real-world scenario testing.

In the real-world scenario testing, four different results were presented. These results were differentiated based on the color correction method used during the setup and whether or not color constancy was also applied. The performance of the object detection algorithm showed a significant improvement when utilizing both CCB-CC and CCB-CC-CC. However, we did not see any significant performance gain while utilizing either GTIB-CC or GTIB-CC-CC, suggesting that utilizing the robot base as a reference over a color checker is not a viable solution without further research being put into this.

We found that performance of the GTIB-CC method also was dependent on the position of the illuminant. When the illuminant illuminated the scene and the ground-truth reference (the robot base) equally, the performance was generally better. However, when the ground-truth reference received a larger impact from the illuminant than the working scene did, this lead to overcompensation and when it received a lower impact this lead to undercompensation. The impact the placement of the illuminant had when utilizing CCB-CC was not as drastic as for GTIB-CC, however, it is still significant.

One of the most intriguing findings was the role of color correction and color constancy in a given setup. While it was hypothesized that combining these two elements would result in a performance increase, the margin of improvement was generally less than anticipated. This could be due to the simultaneous implementation of both methods introducing conflicting signals or adding noise to the system. Interestingly, a more substantial performance gain was witnessed when only color constancy was applied. This suggests that for some applications, relying solely on color constancy might be the most effective method to achieve stable color representation.

However, it is important to note that the results did not fully meet the expectations set at the beginning of the research. We identified performance degradation due to noise in the data as a potential area for improvement, especially for our novel approach, GTIB-CC. One proposed solution was to not use one pixel as input, but instead use an average of a number of nearby pixels. However, predicting the impact of this change on the final performance is challenging and requires further investigation.

In conclusion, while the research did not fully achieve the desired results, it provided valuable insights into the complexities of optimizing color perception in robotics. It highlighted the potential of color constancy as a standalone method for achieving stable color representation and identified areas for further investigation to improve the performance of color correction methods.

7.3 Interpretation of Findings

The findings of this research have some implications for the field of robotics and computer vision, particularly in the context of color perception. The research was guided by the theoretical framework that color is a potent feature in computer vision applications but is susceptible to changes in illumination conditions, which can significantly undermine the effectiveness of computer vision algorithms. Therefore, it is crucial to attempt to neutralize the influence of the illuminant, which is addressed by color correction and color constancy.

The results of our study confirmed that the classical solution for color correction, namely using a color checker is a viable solution for increasing the performance of computer vision applications. While we also introduced a novel approach to color correction that would remove the need for an external tool, we did not achieve the desired level of performance. We believe that through further development this method could be made viable, what this development could entail will be discussed further in the following sections.

We also investigated the impact the position of the light source relative to the ground-truth reference has on the final result of color correction. We found that the position of the light source should be such that the illuminant has a similar level of impact on both the ground-truth reference, be it a color checker or a robot base, as well as the working scene. If this is not fulfilled it can lead to overcompensation or undercompensation in color correction.

We have also confirmed the viability of color constancy in the context of computer vision. We investigated and confirmed that color constancy can be combined with color correction, providing in some cases, a performance gain. However, the margin of improvement was generally less than anticipated, suggesting that implementing both simultaneously might introduce conflicting signals or add noise to the system.

Interestingly, the research found that a more substantial performance gain was witnessed when only color constancy was applied. This suggests that for some applications, relying solely on color constancy might be the most effective method to achieve stable color representation. This finding has significant implications for the development of computer vision applications, suggesting a potential shift in focus towards the development and refinement of color constancy methods.

In conclusion, the findings of this research provide insights into the complexities of optimizing color perception in robotics. They affirmed the effectiveness of classical color correction methods, i.e using a color checker. We also investigated and evaluated a novel approach

to color correction that removes the need for external tool and identified areas for further investigation that improve the performance of this color correction method. The findings also highlight the potential of color constancy as a standalone method for achieving stable color representation. These findings directly address the research questions and contribute to the broader field of study by providing a deeper understanding of the challenges and potential solutions in optimizing color perception in robotics.

7.4 Limitations

While the research conducted in this thesis provides valuable insights into the optimization of color perception in robotics, it is important to acknowledge its limitations. These limitations primarily revolve around the data collection process, the methodology used, and certain aspects that were outside the scope of the thesis but may still impact the interpretation of the findings.

One of the main limitations of the study was the performance degradation due to noise in the data. We identified this as a potential area for improvement, suggesting that future research could focus on developing methods to minimize noise in the data. However, it is important to note that the impact of such changes on the final performance is difficult to predict and requires further investigation.

Another limitation was related to the methodology used for color correction. The research explored the implementation of color correction using external tools, such as a color checker as a reference, and examined a novel approach to color correction that does not rely on external tools but instead utilizes the robot's surrounding environment. While this approach has potential, the results varied significantly across different scenarios, indicating that it may not be universally applicable or reliable.

Furthermore, the research was conducted in the context of a specific use-case: a robot arm with a camera mounted to it. While the findings may be applicable to similar scenarios, they may not be generalize to all robotics or computer vision applications. This limitation is inherent in the scope of the thesis and should be considered when interpreting the findings.

Finally, the research did not fully meet the expectations set at the beginning of the study. We did not achieve the results we hoped for, indicating that there may be other factors at play that were not accounted for in the research. These could include unexplored aspects of color correction and constancy, or other variables related to the robot's environment or the specific implementation of the computer vision algorithms.

In conclusion, while the research provides insights and contributes to the field of robotics and computer vision, these limitations should be taken into account when interpreting the findings and planning future research in this area.

7.5 Future Work

The research conducted in this thesis has opened up several avenues for future work. These areas of future research could continue the study or address limitations identified in the research.

7.5.1 Performance Degradation Due to Noise

One of the main limitations identified in the study was the performance degradation due to noise in the data. There are generally two ways to solve this, either reducing the sensitivity to the noise, or reducing the amount of noise. While both are viable solutions, we will focus on how we can reduce the measurement noise. One such method that could do this is to not use one pixel as input, but instead use an average of a number of nearby pixels as input. This approach could potentially reduce the noise in the data and improve the performance of the color correction methods. However, it is difficult to predict how this would affect the final performance, and as such, needs to be further investigated in future research.

7.5.2 Potential Improvements to GTIB-CC

The Ground-Truth Image Based Color Correction (GTIB-CC) method showed varied results across different scenarios, indicating that it may not be universally applicable or reliable. Future research could focus on improving the GTIB-CC method. This could include researching what an ideal ground-truth image should contain, for example, what should the illumination conditions be when capturing the ground-truth image and what is an ideal scene for this ground-truth image. Another avenue of research with large potential is to improve how the ground-truth image is used, for example developing a method for aligning the ground-truth image with the input image could reduce any measurement errors.

7.5.3 Optimized Integration of Color Correction and Constancy

We found that the integration of color correction and constancy does not always lead to a significant improvement in performance. Future research could focus on exploring ways to better use color constancy and correction at the same time. This could involve developing new methods or refining existing ones to effectively integrate color correction and constancy, potentially leading to improved performance in color perception in robotics.

7.6 Conclusion

In our research, we have made progress in addressing the challenge of optimizing color perception in robotics by integrating color correction and constancy. Our study has shed light on the potential of using a robot base as a ground-truth over a color checker board for color correction and the possible enhancement in color correction results through the application of color constancy. We have also contributed an evaluation of current color correction and color constancy methods in the context of robotic computer vision applications by investigating a real-world situation.

However, our research has also uncovered several limitations and areas for future exploration. One of the main limitations we identified was the degradation in performance due to noise in the data. Future research could focus on developing methods to minimize this noise, potentially improving the performance of color correction methods. Another limitation was

related to the methodology we used for color correction. We explored the implementation of color correction using external tools and a novel approach that does not rely on external tools but instead utilizes the robot's surrounding environment. While this approach has potential, the results varied significantly across different scenarios, indicating that it may not be universally applicable or reliable.

Furthermore, our research was conducted in the context of a specific use-case: a robot arm with a camera mounted to it. While our findings may be applicable to similar scenarios, they may not generalize to all robotics or computer vision applications. This limitation is inherent in the scope of our thesis and should be considered when interpreting our findings.

Finally, our research did not fully meet the expectations we set at the beginning of the study. We did not achieve the results we hoped for, indicating that there may be other factors at play that we did not account for in our research. These could include unexplored aspects of color correction and constancy, or other variables related to the robot's environment or the specific implementation of the computer vision algorithms.

In conclusion, while our research provides insights and contributes to the field of robotics and computer vision, these limitations should be taken into account when interpreting our findings and planning future research in this area. Future research could focus on exploring ways to better use color constancy and correction simultaneously, potentially leading to improved performance in color perception in robotics.

References

[1] Mahmoud Afifi et al. “When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA: IEEE, June 2019, pp. 1535–1544. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00163. (Visited on 05/30/2023).

[2] Vivek Agarwal et al. “An Overview of Color Constancy Algorithms”. In: *Journal of Pattern Recognition Research* 1.1 (2006), pp. 42–54. ISSN: 1558884X. DOI: 10.13176/11.9. (Visited on 03/30/2023).

[3] Kendall E. Atkinson. *An Introduction to Numerical Analysis*. 2nd ed. New York: Wiley, 1989. 693 pp. ISBN: 978-0-471-62489-9.

[4] *Auto White Balance (AWB)*. URL: https://www.kaggle.com/datasets/tenxengineers/auto-white-balance-awb (visited on 05/12/2023).

[5] Inês Bramão et al. “The Role of Color Information on Object Recognition: A Review and Meta-Analysis”. In: *Acta Psychologica* 138.1 (Sept. 1, 2011), pp. 244–253. ISSN: 0001-6918. DOI: 10.1016/j.actpsy.2011.06.010. (Visited on 03/30/2023).

[6] Vlad C Cardei and Brian Funt. “Committee-Based Color Constancy”. In: (Mar. 7, 2000).

[7] Jonathan Cepeda-Negrete and Raul E Sanchez-Yanez. “Color Constancy Algorithms in Practice”. In: (2011). DOI: 10.13140/RG.2.1.1956.6569. (Visited on 04/03/2023).

[8] Dongliang Cheng et al. “Effective Learning-Based Illuminant Estimation Using Simple Features”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015, pp. 1000–1008. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Cheng_Effective_Learning-Based_Illuminant_2015_CVPR_paper.html (visited on 05/10/2023).

[9] Vien Cheung et al. “A Comparative Study of the Characterisation of Colour Cameras by Means of Neural Networks and Polynomial Transforms”. In: *Coloration Technology* 120.1 (Jan. 2004), pp. 19–25. ISSN: 1472-3581, 1478-4408. DOI: 10.1111/j.1478-4408.2004.tb00201.x. (Visited on 05/03/2023).

[10] *Chromaticity*. In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Chromaticity&oldid=1157063583 (visited on 06/02/2023).

[11] *Colour-Science/Colour*. colour-science. URL: https://github.com/colour-science/colour (visited on 05/26/2023).

[12] *Colour.Colour_correction — Colour 0.4.2 Documentation*. URL: https://colour.readthedocs.io/en/develop/generated/colour.colour_correction.html (visited on 05/26/2023).

[13] Kenneth Dawson-Howe. *A Practical Introduction to Computer Vision with OpenCV*. Chichester, West Sussex, United Kingdon ; Hoboken, N.J: John Wiley & Sons., Inc, 2014. 217 pp. ISBN: 978-1-118-84845-6.

[14] *Depth Camera D405 – Intel® RealSense™ Depth and Tracking Cameras*. URL: https://www.intelrealsense.com/depth-camera-d405/ (visited on 03/29/2023).

[15] Marc Ebner. "Color Constancy". In: *Computer Vision: A Reference Guide*. Ed. by Katsushi Ikeuchi. Boston, MA: Springer US, 2014, pp. 109–116. ISBN: 978-0-387-31439-6. DOI: 10.1007/978-0-387-31439-6_454. (Visited on 03/30/2023).

[16] *Edge Detection - Computer Vision - Computer Science Field Guide*. URL: https://www.csfieldguide.org.nz/en/chapters/computer-vision/edge-detection/ (visited on 03/30/2023).

[17] G. Finlayson and S. Hordley. "Improving Gamut Mapping Color Constancy". In: *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society* 9.10 (2000), pp. 1774–1783. ISSN: 1057-7149. DOI: 10.1109/83.869188. pmid: 18262915.

[18] Graham D. Finlayson. "Colour and Illumination in Computer Vision". In: *Interface Focus* 8.4 (Aug. 6, 2018), p. 8. ISSN: 2042-8898, 2042-8901. DOI: 10.1098/rsfs.2018.0008. (Visited on 03/29/2023).

[19] Graham D. Finlayson, Michal Mackiewicz, and Anya Hurlbert. "Color Correction Using Root-Polynomial Regression". In: *IEEE Transactions on Image Processing* 24.5 (May 2015), pp. 1460–1470. ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2015.2405336. (Visited on 05/03/2023).

[20] *GoFa CRB 15000 Collaborative Robot | ABB Robotics*. Robotics. URL: https://new.abb.com/products/robotics/robots/collaborative-robots/crb-15000 (visited on 03/29/2023).

[21] *How to Achieve? | Color Constancy*. URL: http://colorconstancy.com/color-constancy/how/./index.html (visited on 03/30/2023).

[22] Md Akmol Hussain, Akbar Sheikh-Akbari, and Iosif Mporas. "Colour Constancy for Image of Non-Uniformly Lit Scenes". In: *Sensors* 19.10 (10 Jan. 2019), p. 2242. ISSN: 1424-8220. DOI: 10.3390/s19102242. (Visited on 03/30/2023).

[23] *Introduction (Image Processing Toolbox)*. URL: http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/images/intro8.html (visited on 03/30/2023).

[24] Sivagami K.P. et al. "EDGE DETECTION USING MULTISPECTRAL THRESHOLDING". In: *ICTACT Journal on Image and Video Processing* 06 (May 1, 2016), pp. 1262–1272. DOI: 10.21917/ijivp.2016.0184.

[25] Rok Kreslin et al. “Linear Chromatic Adaptation Transform Based on Delaunay Triangulation”. In: *Mathematical Problems in Engineering* 2014 (2014), pp. 1–9. ISSN: 1024-123X, 1563-5147. DOI: 10.1155/2014/760123. (Visited on 05/03/2023).

[26] Shiming Lai et al. “Fast and Robust Color Constancy Algorithm Based on Grey Block-Differencing Hypothesis”. In: *Optical Review* 20 (July 1, 2013). DOI: 10.1007/s10043-013-0062-x.

[27] Wei-Yin Loh. “Classification and Regression Trees”. In: ().

[28] *Luminance*. In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Luminance&oldid=1152018062 (visited on 06/02/2023).

[29] *OpenCV: Additional Photo Processing Algorithms*. URL: https://docs.opencv.org/4.x/de/daa/group__xphoto.html (visited on 05/26/2023).

[30] *OpenCV: Macbeth Chart Module*. URL: https://docs.opencv.org/4.x/dd/d19/group__mcc.html (visited on 05/26/2023).

[31] *Raspberry - Wikipedia*. URL: https://en.wikipedia.org/wiki/Raspberry (visited on 05/12/2023).

[32] *ROS 2 Documentation — ROS 2 Documentation: Galactic Documentation*. URL: https://docs.ros.org/en/galactic/index.html (visited on 10/30/2023).

[33] *Spectral Power Distribution (SPD)*. GTI Graphic Technology Inc. URL: https://www.gtilite.com/color-viewing-terms/spectral-power-distribution-spd/ (visited on 03/30/2023).

[34] Robby T. Tan, Ko Nishino, and Katsushi Ikeuchi. “Color Constancy through Inverse-Intensity Chromaticity Space”. In: *Journal of the Optical Society of America A* 21.3 (Mar. 1, 2004), p. 321. ISSN: 1084-7529, 1520-8532. DOI: 10.1364/JOSAA.21.000321. (Visited on 03/29/2023).

[35] Aslak Tveito et al. “Parameter Estimation and Inverse Problems”. In: *Elements of Scientific Computing*. Vol. 7. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 411–421. ISBN: 978-3-642-11298-0 978-3-642-11299-7. DOI: 10.1007/978-3-642-11299-7_9. (Visited on 06/01/2023).

[36] *Vandermonde Matrix*. In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Vandermonde_matrix&oldid=1152711871 (visited on 05/03/2023).

[37] Xiaohong Yan et al. “A Novel Biologically-Inspired Method for Underwater Image Enhancement”. In: *Signal Processing: Image Communication* 104 (May 1, 2022), p. 116670. ISSN: 0923-5965. DOI: 10.1016/j.image.2022.116670. (Visited on 03/30/2023).

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-10-26

EXAMENSARBETE Optimizing Color Perception in Robotics: Integrating Color Correction and Constancy

STUDENT Erik Folkesson
HANDLEDARE Volker Krueger (LTH)
EXAMINATOR Jacek Malec (LTH)

Optimizing Color Perception in Robotics

POPULÄRVETENSKAPLIG SAMMANFATTNING **Erik Folkesson**

This thesis delves into the complexities of color perception in robotics, presenting a unique approach for integrating color correction and color constancy. However, real-world testing reveals limitations.

Achieving accurate color perception in robots is complex due to variables like lighting and operating conditions. Traditional techniques often require specialized calibration or controlled lighting, impractical for robots in diverse, real-world settings. This research develops a new plug-and-play method for color correction and constancy for the Robot Operating System 2 (ROS2), aiming for autonomous operation without external tools. Extensive experimentation was conducted, evaluating various algorithms using metrics like Mean Squared Error (MSE).

A comparison of the original and corrected images against the ground-truth image is presented on the next page, aiming for consistency between the corrected input and ground-truth images.

In controlled environments, the methodology showed promise for broad applications in robotics where accurate color perception is crucial. However, the study faced challenges in real-world scenarios, with the performance of the color correction algorithms deteriorating under fluctuating lighting conditions, limiting its applicability in dynamic settings.

These limitations underline the complexity of the problem and open up avenues for further study, focusing on improving the robustness of color perception algorithms under varying conditions. This research serves as a step towards accurate color perception in robotics, emphasizing the complexities that remain.