

# TDK-dolgozat

Sziksz Melinda

# Color and shade identification for multiple clothing items simultaneously for an application designed for the visually impaired

2024. 11. 19.

## **Intézmény:**

Pázmány Péter Katolikus Egyetem  
Információs Technológiai és Bionikai Kar

## **Szerző:**

Szikszo Melinda  
mérnökinformatikus BSc, 7. félév

## **Témavezetők:**

Dr. Gelencsér-Horváth Anna, egyetemi adjunktus  
Dr. Karacs Kristóf, egyetemi docens

# Abstract

This thesis presents a comprehensive study on the identification and clustering of colors and shades for multiple clothing items, specifically aimed at assisting visually impaired individuals. The main objective is to develop a practical solution that can accurately identify colors from clothing items simultaneously, rather than analyzing them individually, making it significantly faster and more useful in real-world scenarios. The system is designed to help visually impaired individuals overcome challenges related to color identification and selection, which are crucial in daily activities such as choosing matching clothes or organizing items by color. This work is motivated by the *Colors* task of the Cybathlon competition, which serves as a testing ground for assistive technologies that empower visually impaired individuals.

The proposed solution leverages both classical image processing techniques and advanced computer vision algorithms. Initially, I explored traditional methods such as K-means clustering, SLIC superpixel segmentation, adaptive thresholding, and Otsu's thresholding for clothing segmentation, but these proved insufficient due to inconsistencies caused by lighting variations and shadows. As a result, a more robust solution was developed using the Fast Segment Anything Model (FastSAM), which improved accuracy and efficiency in segmenting the six clothing items present in each frame.

The system employs a combination of RGB and depth images captured using a smartphone's LiDAR sensor to identify regions of interest while excluding background interference. To improve the accuracy of color identification, the implementation utilizes a color reference card, which includes a range of pre-defined colors, allowing for reliable calibration across various lighting conditions. The colors are extracted from the reference card through a homographic transformation, which corrects distortions caused by perspective, ensuring that the colors are sampled accurately for subsequent analysis.

The color analysis module employs the CIEDE2000 color difference formula to determine the distance between sampled color values from clothing items and the reference colors, ensuring perceptual uniformity. The Hungarian algorithm is applied for optimal

assignment when there is ambiguity in color identification. The system features an auditory feedback mechanism to communicate color information to the user in an accessible format, facilitating ease of use for visually impaired individuals.

To evaluate the system's effectiveness, I conducted a series of tests using videos captured in real-world conditions, replicating the environment of the Cybathlon competition. The results showed that the new solution significantly reduced task completion time compared to previous implementation, achieving an average of 33 seconds per task compared to the previous 107 seconds. The system achieved high accuracy in garment order detection, reaching 92.98% with a memory setting of 6. The accuracy of correctly detecting the color of a single clothing item reached 97% with this memory setting.

This work contributes to the development of assistive technologies that improve the autonomy of visually impaired individuals. Future improvements may include enhancing the robustness of color differentiation through machine learning techniques and adapting the system to handle a broader range of garments and scenarios beyond the controlled competition environment. The ultimate goal is to create a versatile and reliable color identification tool that can be used in daily life, enhancing the independence and quality of life for visually impaired users.

# Contents

|                                                                     |           |
|---------------------------------------------------------------------|-----------|
| <b>Abstract</b>                                                     | <b>ii</b> |
| <b>1 Introduction</b>                                               | <b>1</b>  |
| 1.1 Motivation . . . . .                                            | 1         |
| 1.2 Structure of the thesis . . . . .                               | 2         |
| <b>2 Related work and background</b>                                | <b>4</b>  |
| 2.1 Color spaces and the CIEDE2000 formula . . . . .                | 4         |
| 2.2 Image processing and segmentation techniques . . . . .          | 6         |
| 2.3 Computer vision algorithms . . . . .                            | 11        |
| 2.4 The previous implementation of the <i>Colors</i> task . . . . . | 13        |
| <b>3 Methods</b>                                                    | <b>16</b> |
| 3.1 Defining the region of interest . . . . .                       | 17        |
| 3.2 Clothing segmentation and mask extraction . . . . .             | 22        |
| 3.3 Color analysis and auditory feedback . . . . .                  | 26        |
| 3.3.1 Color analysis . . . . .                                      | 26        |
| 3.3.2 Auditory feedback . . . . .                                   | 31        |
| 3.4 Implementation frameworks . . . . .                             | 32        |
| <b>4 Results</b>                                                    | <b>33</b> |
| <b>5 Conclusion and future work</b>                                 | <b>37</b> |
| 5.1 Project summary . . . . .                                       | 37        |
| 5.2 Discussion and future work . . . . .                            | 38        |
| <b>Bibliography</b>                                                 | <b>42</b> |

# Chapter 1

## Introduction

### 1.1. Motivation

For individuals with visual impairments, identifying and sorting items by color poses a significant challenge. Colors cannot be perceived through touch, sound, or any other sense, making this seemingly simple task impossible without assistance. Yet, color identification is an essential part of daily life, such as when selecting matching clothes or organizing items based on appearance. This reality underscores the need for technology that can empower the visually impaired to overcome these challenges, thereby enabling them to live more independently.

The Cybathlon, organized by ETH Zurich, serves as a platform that encourages teams from around the world to develop assistive technologies for everyday use. [1] These competitions bring together technology developers and individuals with disabilities—known as pilots—to solve practical daily challenges using their latest innovations. By participating in the Cybathlon, our goal is not only to compete but also to drive the advancement of assistive technologies that can have a real and positive impact on people’s lives.

The Cybathlon contest’s *Colours* task directly addresses the challenges of color identification faced by individuals with visual impairments. The task requires pilots to identify sets of different colors and sort them according to their gradations. Specifically, pilots must work with two colors chosen from a pool of five predefined colors, each presented in three distinct gradations. The two colors in each race may be any of the following pairs: red & green, red & cyan, red & blue, magenta & green, magenta & cyan, or green & blue. These colors and their gradations are considered easily distinguishable by people without visual impairments. The clothes hangers with the fabric samples are initially

placed randomly on a clothesline, and the pilot must sort them correctly by grasping and positioning them. [1]

Sorting clothes based on color may seem trivial for someone who can see, but for visually impaired individuals, it presents numerous difficulties. When using a camera, the challenges are even more pronounced, as a blind person has no visual reference to ensure that the camera is focused on the correct item or to verify that there are no extraneous objects in the frame that could interfere with the analysis. Therefore, the solution must be both user-friendly and reliable in real-world settings. It is not enough for developers to simply test the system under controlled conditions; it must be designed with the user experience as a central focus. In addition, efficiency is crucial, which becomes even more critical at Cybathlon, as there are only eight minutes to complete all ten tasks. This means that the color-sorting task must be completed in one minute or less.

This time constraint demands a solution that is both quick and robust, capable of handling varying and unpredictable lighting conditions during the competition. One of the most challenging aspects of this task is distinguishing between different shades of the same color. Differentiating between distinct colors like red and blue can be relatively straightforward, but identifying different shades is significantly more complex — even people with healthy vision can struggle with this under changing lighting conditions. Therefore, our system must be able to perform accurately across a range of lighting environments to ensure consistent results. The solution must be resilient enough to handle subtle variations and still correctly identify the intended shade.

I joined the EyeRider team to help address this challenge. The team has been developing a mobile application called LetSee, available on the App Store and Google Play Store, which helps individuals with visual impairments in a variety of tasks. Our current project continues in this spirit, utilizing only a smartphone to provide a solution that can assist not only during the competition but also in everyday scenarios. By focusing on smartphone technology, I aim to create a solution that is both accessible and practical, helping the visually impaired to independently tackle the challenges of color identification.

## 1.2. Structure of the thesis

**Chapter 2** contains the related work, explaining the research I conducted in image and color processing. It reviews existing literature, detailing methods and techniques developed by others. Most of these methods are forming a foundation for my solution of the *Colors* task.

**Chapter 3** presents the technical details of the solution for the *Colors* task. It explains how the program and the used methods work. This chapter covers the implementation process, including algorithms, software tools, and challenges encountered. Each component is discussed, providing insight into design choices and their reasoning.

**Chapter 4** discusses the test results, demonstrating the effectiveness of the solution in identifying colors. It includes quantitative and qualitative analysis of performance, highlighting metrics like accuracy, speed, and reliability. The results are analyzed to assess strengths and limitations, offering a clear picture of its effectiveness.

**Chapter 5** contains the conclusion, summarizing project goals and outlining future work. It reflects on challenges faced, successes, and opportunities for development. Broader implications, such as potential impact on daily lives and adapting the solution to related challenges, are also discussed. The chapter concludes by identifying areas for improvement and suggesting future research directions.

The code of the program will be available in the following repository: <https://github.com/Systemense0/Bsc-Thesis.git>

## Chapter 2

# Related work and background

### 2.1. Color spaces and the CIEDE2000 formula

The purpose of a color model, also known as a color space or color system, is to standardize the specification of colors, enabling a consistent interpretation of color data. A color model defines both a coordinate system and a subspace within it, where each point corresponds to a unique color. Common color spaces in digital imaging include RGB (Red, Green, Blue), BGR (Blue, Green, Red), HSV (Hue, Saturation, Value), and CIE LAB. Each model serves distinct applications, tailored either toward hardware requirements or representing color in ways that align with human perception [2].

In the RGB color model, colors are represented by combining the three primary colors—red, green, and blue—based on a Cartesian coordinate system. This model forms a color cube where each axis corresponds to one primary color, with intensities normalized within the range  $[0, 255]$ . Black is at the origin  $(0,0,0)$ , while white is at the point  $(255, 255, 255)$ , forming a grayscale diagonal across the cube. RGB is widely used in displays and digital cameras, where a 24-bit depth allows for  $2^{24} = 16,777,216$  color variations. The BGR model, primarily used in OpenCV, reorders these channels as Blue, Green, and Red for specific hardware compatibility [2].

The HSV color model is designed to align more closely with human color perception, making it particularly useful for tasks that require intuitive color manipulation. In this model, Hue represents the color type (e.g., red or blue), typically measured in degrees from 0 to 360. Saturation denotes color purity, ranging from 0 (gray) to 100% (pure color), while Value corresponds to brightness, from 0 (black) to full intensity. The HSV model effectively decouples color information from intensity, which can simplify tasks like filtering and object detection [2].

Figure 2.1 demonstrates the relationship between RGB and HSI color spaces, focusing on the concepts of intensity, saturation, and hue. In the RGB color cube, black is positioned at the bottom and white at the top, creating a vertical intensity axis. The intensity of a color is determined by the intersection of a plane containing that color with the intensity axis, yielding a value between 0 and 1. Saturation increases as the distance from the intensity axis grows, with points along the axis representing pure gray (zero saturation). Hue is defined by rotating a plane around the intensity axis, meaning all points within that plane share the same hue. This framework allows for the conversion of RGB values into the HSI color space, enabling the direct calculation of hue, saturation, and intensity from the RGB cube. [2]

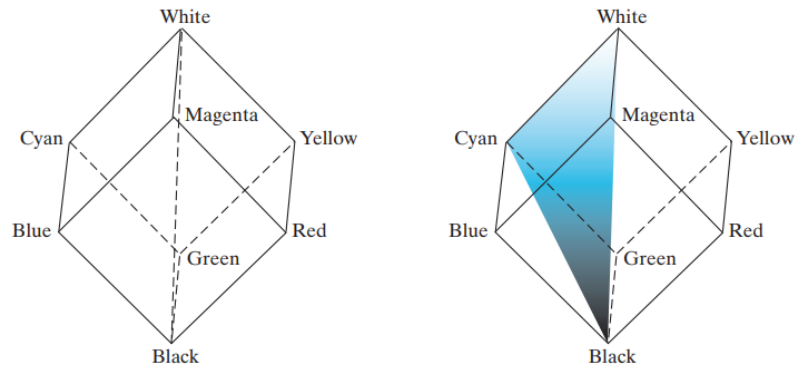


Figure 2.1: Conceptual relationships between the RGB and HSV color models. [2]

The CIE LAB color model, established by the International Commission on Illumination (CIE), is designed to provide perceptual uniformity, aiming for the distances between points in the color space to correspond closely to perceived color differences. This device-independent model is widely adopted for color calibration and quality control across industries due to its three-dimensional structure:  $L^*$  for lightness (ranging from 0, representing black, to 100, representing white),  $a^*$  for the green-red axis, and  $b^*$  for the blue-yellow axis. This structure enables the representation of the entire visible color spectrum, making it a preferred choice for applications requiring precise color consistency across different devices.[3]

To quantify color differences within this model, particularly in applications where color matching is crucial, various color difference formulas have been developed. Initial attempts, such as the simple Euclidean distance in CIELAB, failed to align well with human perception. Consequently, more refined formulas like the CMC, BFD, and CIE94 emerged, each introducing unique weighting and scaling functions to better capture perceptual nuances by separately considering differences in lightness, chroma, and

hue. However, these methods also had limitations, including an asymmetry issue where color difference results could vary based on which color was set as the standard versus the sample.[3]

The CIEDE2000 formula, introduced as a more advanced alternative, addresses these limitations. It builds upon the features of previous models but adds critical adjustments for non-uniformities in the CIELAB space by incorporating hue-chroma interaction terms and applying additional scaling factors for lightness, chroma, and hue differences. The CIEDE2000 color difference formula is expressed as follows:

$$\Delta E_{00} = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \frac{\Delta C'}{k_C S_C} \frac{\Delta H'}{k_H S_H}}$$

where:

- $\Delta L'$ ,  $\Delta C'$ , and  $\Delta H'$  represent differences in lightness, chroma, and hue.
- $S_L$ ,  $S_C$ , and  $S_H$  are weighting functions for lightness, chroma, and hue, respectively.
- $k_L$ ,  $k_C$ , and  $k_H$  are parametric factors, typically set to 1 under standard conditions.
- $R_T$  is an additional term that considers the interaction between chroma and hue differences, correcting perceptual inconsistencies, especially in hues where human sensitivity varies, such as blues and purples.

A distinctive improvement in the CIEDE2000 formula is the hue-chroma interaction term  $R_T$ , which compensates for perceptual discrepancies in regions of the color space with higher perceptual sensitivity. Additionally, the parametric factors  $k_L$ ,  $k_C$ , and  $k_H$  allow for adjustments based on specific viewing conditions, enhancing the robustness and adaptability of the CIEDE2000 formula for applications requiring high color fidelity.[3]

## 2.2. Image processing and segmentation techniques

### Contrast Limited Adaptive Histogram Equalization (CLAHE)

Enhancing image contrast is crucial in digital image processing, particularly for images with poor visibility of details due to limited contrast. One of the foundational methods in this domain is histogram equalization, which redistributes the intensities in an image to achieve a more uniform spread across the grayscale. By adjusting the image's intensity distribution, histogram equalization aims to make important features more distinguishable. However, this global approach often falls short when applied to images with locally

varying brightness or contrast, as it may overemphasize noise in regions where intensity variation is minimal [2].

To address these issues, Adaptive Histogram Equalization (AHE) was introduced, dividing an image into smaller regions and applying histogram equalization locally within each region. This technique improves local contrast, allowing details in different regions to be better distinguished. Nevertheless, AHE can significantly amplify noise, especially in homogeneous areas where pixel intensities are closely clustered. This limitation led to the development of Contrast Limited Adaptive Histogram Equalization (CLAHE), an enhancement to AHE that effectively controls noise amplification while enhancing contrast [4].

CLAHE limits the contrast enhancement by introducing a clip limit for the histogram in each local region. By capping the maximum height of the histogram bins, CLAHE avoids excessive contrast enhancement in areas with high pixel uniformity, which often corresponds to noise. Once the histogram reaches the clip limit, excess values are redistributed across the histogram, ensuring that contrast enhancement is balanced without overemphasizing noise. The degree of contrast limitation is controlled by the clip limit parameter: a lower clip limit reduces noise visibility by limiting contrast enhancement in homogeneous areas, while a higher limit approaches standard AHE and allows for more pronounced contrast enhancement. The effectiveness of CLAHE makes it well-suited for applications where enhancing local details is essential without compromising image quality with noise [4].

## Image moments and moment invariants

Image moments are fundamental tools in image analysis, providing a way to capture the distribution of pixel intensities and enabling recognition of shapes independent of transformations like translation, scaling, and rotation. Moments effectively summarize spatial characteristics of an image, and Hu's work on moment invariants introduced descriptors that remain unchanged under transformations, facilitating robust pattern recognition [5].

In a two-dimensional image  $f(x, y)$ , the raw moments  $m_{pq}$  are defined as:

$$m_{pq} = \int \int x^p y^q f(x, y) dx dy$$

where the indices  $p$  and  $q$  determine the order of the moment.

Key moments in image analysis include  $m_{00}$ ,  $m_{10}$ ,  $m_{01}$ , and  $m_{11}$ , each offering specific insights into an image's structure. The moment  $m_{00}$  represents the total intensity of the image, essentially summing all pixel values, and is often interpreted as the “mass” of the image. Moments  $m_{10}$  and  $m_{01}$  are used to determine the centroid, or center of mass, of the image. The centroid coordinates, calculated as  $\bar{x} = \frac{m_{10}}{m_{00}}$  and  $\bar{y} = \frac{m_{01}}{m_{00}}$ , serve as a reference point for defining moments relative to the center of the image. Additionally,  $m_{11}$  captures information on the distribution of intensity in relation to both the  $x$  and  $y$  axes, aiding in computing the image's orientation and proving useful in constructing rotation-invariant features.

To ensure translation invariance, central moments  $\mu_{pq}$  are calculated relative to the centroid and defined as:

$$\mu_{pq} = \int \int (x - \bar{x})^p (y - \bar{y})^q f(x, y) dx dy.$$

By building on these central moments, Hu derived moment invariants that are consistent across translation, scaling, and rotation. These invariant features are particularly valuable in applications like optical character recognition and shape classification, where the recognition of patterns must be reliable, regardless of their position, size, or orientation [5].

## Image rectification

Image rectification is a transformation technique used to adjust images taken from different perspectives, aligning them as if viewed from a single, consistent viewpoint. This process is also referred to as perspective transformation or homography [6].

In mathematical terms, rectification operates on homogeneous coordinates, where each point  $\tilde{x}$  in an image is represented by a coordinate vector that includes an additional scaling component. This process is governed by a transformation matrix  $\tilde{H}$ , a  $3 \times 3$  matrix in 2D, such that:

$$\tilde{x}' = \tilde{H} \tilde{x}$$

where  $\tilde{H}$  is defined up to a scale factor, meaning that any two matrices that differ only by a scaling factor are equivalent. After the transformation, the resulting homogeneous coordinates  $\tilde{x}'$  must be normalized to obtain inhomogeneous coordinates  $(x', y')$ , which represent the actual transformed point in the image. The normalized coordinates are

computed as:

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}}, \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}$$

where  $h_{ij}$  are the elements of the matrix  $\tilde{H}$ .

One of the fundamental properties of perspective transformations is that they preserve straight lines; that is, lines remain straight after the transformation is applied. This property makes homographies ideal for applications where maintaining linear structures in the image is important. [6]

## Segmentation techniques

Image segmentation is the process of dividing an image into coherent regions that represent meaningful structures or objects. This process is essential in computer vision applications where precise identification and isolation of regions are required, such as in medical imaging, autonomous driving, and object detection. Segmentation simplifies analysis by focusing on specific areas of interest, facilitating further processing and interpretation [6].

Segmentation techniques are generally classified into threshold-based methods, clustering approaches, and region grouping strategies, each offering unique advantages for different types of images. Thresholding is one of the most straightforward methods, classifying pixels as foreground or background based on their intensity values. While global thresholding applies a single threshold across the image, adaptive thresholding determines a unique threshold  $T(x, y)$  for each pixel based on its local neighborhood, adjusting dynamically to account for changes in brightness across the image. This local threshold  $T(x, y)$  is typically computed as the mean or median intensity in a small window around the pixel, with an offset constant  $C$  for refinement:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T(x, y) \\ 0 & \text{if } f(x, y) \leq T(x, y) \end{cases}$$

where  $f(x, y)$  is the pixel intensity at location  $(x, y)$ , and  $T(x, y)$  is calculated as:

$$T(x, y) = \frac{1}{|N(x, y)|} \sum_{(i, j) \in N(x, y)} f(i, j) - C$$

with  $N(x, y)$  representing the local neighborhood. This adaptive approach is particularly useful for images with varying backgrounds, as it adjusts to local intensity distributions and improves segmentation accuracy [2].

Otsu’s thresholding [7], a global method, determines an optimal threshold by maximizing the variance between two classes, foreground and background. By analyzing the between-class variance  $\sigma_B^2$  as a function of different threshold values, Otsu’s method identifies the threshold  $k$  that best separates these classes:

$$h(k) = \frac{\sigma_B^2(k)}{\sigma_G^2}$$

where  $\sigma_G^2$  is the global variance. This technique is effective in images with bimodal intensity histograms, where clear distinctions exist between object and background regions [2].

Clustering-based segmentation methods, such as k-means [8], [9] and mean-shift [10], classify pixels into groups based on feature similarity, such as intensity or color. In k-means clustering, each pixel is assigned to the nearest cluster mean, iteratively refining the means to minimize within-cluster variance. Although effective for distinguishing regions with distinct colors or intensities, k-means requires the number of clusters to be predefined, which may limit flexibility in complex scenes. Mean-shift, on the other hand, is a non-parametric clustering technique that dynamically identifies clusters by locating peaks in the feature space density, grouping pixels that converge to the same peak. This method excels in scenarios with complex, multi-modal distributions and is commonly applied to segment images based on both spatial and color information. By analyzing clusters in this multi-dimensional space, mean-shift preserves spatial coherence while effectively capturing regions of similar appearance [6].

A modern approach to segmentation, superpixel generation [11], divides the image into compact, visually meaningful regions called superpixels. Unlike traditional pixel-based methods, superpixels offer a mid-level representation, grouping pixels into larger, coherent regions based on color, texture, or intensity. This reduces the complexity of the image data, preserving important boundaries while eliminating unnecessary detail. Superpixel-based segmentation improves computational efficiency and segmentation quality, especially in tasks that require rapid processing, such as robotic vision or large-scale image databases. Superpixels are also advantageous for hierarchical segmentation techniques, where they can serve as a basis for more refined segmentation layers, balancing detail preservation with computational efficiency [6].

A recent advancement in image segmentation is the application of deep learning, where neural networks have significantly improved segmentation accuracy, speed, and flexibility. Convolutional Neural Networks (CNNs) form the backbone of most modern segmentation architectures, allowing models to learn complex spatial features directly from data. [6]

One such example is the FastSAMPredictor [12], part of the Ultralytics YOLO framework [13], which is designed to streamline segmentation by leveraging the Fast Segment Anything Model (FastSAM). The FastSAMPredictor extends the SegmentationPredictor class and introduces specialized post-processing, including mask prediction and non-maximum suppression (NMS). [14] This tailored processing is optimized for single-class segmentation, allowing for rapid and efficient segmentation of objects within images. The predictor builds upon YOLOv8-seg [15], a segmentation model within the YOLO family known for its speed and accuracy in object detection, and adapts it specifically for segmentation tasks.

FastSAM itself addresses the computational demands of the Segment Anything Model (SAM) by dividing the task into two distinct stages. In the first stage, it performs all-instance segmentation using YOLOv8-seg, generating a preliminary set of segmented regions. In the second stage, FastSAM filters these regions based on user interaction prompts, selecting only the areas of interest. This two-stage approach not only reduces computational overhead but also allows for greater user control, as users can specify regions of interest without manually outlining each segment.

## 2.3. Computer vision algorithms

Two critical algorithms in this project are the Hungarian Algorithm [16], which addresses assignment problems, and RANSAC Regression [17], which provides robust parameter estimation in noisy datasets. This section describes these algorithms in detail as they take crucial part in my work.

### Hungarian algorithm

The Hungarian Algorithm, introduced by Kuhn [16], provides an efficient approach for solving the assignment problem, which involves finding the minimum-cost perfect matching between two equally sized sets. The algorithm has a time complexity of  $O(n^3)$  and guarantees an optimal solution. Given a cost matrix  $C = [c_{ij}] \in \mathbb{R}^{n \times n}$ , where  $c_{ij}$

represents the cost of assigning agent  $i$  to task  $j$ , the objective is to identify a permutation  $\sigma$  of  $\{1, 2, \dots, n\}$  that minimizes the total assignment cost:

$$\min_{\sigma} \sum_{i=1}^n c_{i\sigma(i)}. \quad (2.1)$$

The algorithm achieves this by transforming the cost matrix in a sequence of steps that highlight zero-cost assignments. The first step is a row reduction, where the smallest element in each row is subtracted from every element of that row:

$$c'_{ij} = c_{ij} - \min_j c_{ij}. \quad (2.2)$$

This is followed by a column reduction, where the smallest element in each column is subtracted from every element in that column:

$$c''_{ij} = c'_{ij} - \min_i c'_{ij}. \quad (2.3)$$

The next step is to assign zeros in a way that each agent is matched to one task. If a complete assignment is achieved, the solution is optimal. Otherwise, the matrix undergoes an adjustment step, where zeros are covered with a minimal number of lines, and uncovered elements are modified to create additional zeros. This process is repeated until a complete assignment is achieved.

## RANSAC regression

RANSAC (Random Sample Consensus), developed by Fischler and Bolles [17], is a robust iterative algorithm used to estimate model parameters in data containing both inliers (points that fit the model) and outliers (points that do not). The primary objective is to fit the model to the inliers while ignoring the outliers, providing a more accurate representation of the underlying data.

The algorithm proceeds by selecting the minimum number of data points required to estimate the model parameters. For instance, in the case of fitting a line to 2D points, RANSAC selects two points per iteration to define a line. A model is then fit to this subset, and the entire dataset is checked to see how many points fall within a predefined error threshold,  $t$ . Points within this threshold are considered inliers, forming a consensus set.

The model’s quality is determined by the size of the consensus set, or the number of inliers. If this set is larger than any previously found, it becomes the current best model. This process repeats for a fixed number of iterations  $k$ , calculated to ensure a high probability of finding a subset with only inliers at least once. The formula for  $k$  is:

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}, \quad (2.4)$$

where  $n$  is the number of points required to estimate the model,  $w$  is the probability of selecting an inlier, and  $p$  is the confidence level of obtaining a model with only inliers.

After all iterations, the model with the largest consensus set is chosen as the best-fit model, which can be further refined by recalculating it using all inliers in the consensus set. RANSAC is effective in separating inliers from outliers, allowing for robust model estimation even in noisy datasets. The algorithm’s performance is influenced by parameters  $t$ ,  $d$ , and  $k$ , which determine the inlier threshold, the number of necessary inliers, and the confidence level in identifying a suitable model.

## 2.4. The previous implementation of the *Colors* task

Before developing a more effective methodology for the task, it was necessary to familiarize with the previous implementation [18].

The system used a physical color card to read the colors of the clothing items one by one. The card contained all the colors and shades required for the *Colors* task, with a hole in the center where the fabric of the clothing could be seen. The color of the fabric was then compared to the reference colors on the card. The purpose of the color card was to ensure that, even under different lighting conditions, the system could accurately determine which color and shade the fabric most closely resembled.

### Detection of the color card

To detect the color card in the camera’s view, the system used ArUco markers [19], which were placed at the four corners of the card. ArUco markers are square fiducial markers that consist of a black square border with a unique binary code pattern inside, allowing for robust detection and identification of individual markers even in poor lighting conditions or when partially occluded [20]. Once the ArUco markers were detected, their positions were used to identify the corners of the card within the image.

After locating the card, a homography transformation was applied to map the four corners of the card to a rectangular shape. This transformation corrected any distortions

caused by the perspective of the camera, ensuring that the card and the colors on it were viewed as if from directly above, regardless of the actual camera angle. By applying this transformation, the system could more accurately assess the color values from the card.

### **Clustering**

Once the fabric's section visible through the hole in the card was identified, multiple sample points were taken. To manage color variations due to inconsistent lighting or shadows, a clustering approach based on color distance was implemented. The process begins by evaluating each detected color value. Each cluster tracks the sum of its color values, the count of colors in the cluster, and the average color, which serves as the cluster's centroid.

As new color values are evaluated, they are either added to an existing cluster if their distance from the centroid is below a predefined threshold, or a new cluster is created for them. This ensures that only similar colors are grouped together while significantly different values form separate clusters.

Once all detected color values have been assigned to clusters, the two largest clusters are compared. If the difference in their sizes is below a specified threshold, the clustering result is considered inconclusive. However, if one cluster is clearly larger, its centroid is selected as the representative color value.

### **Determining the color of the clothing**

After selecting the representative color value from the largest cluster, the system proceeds to determine the color of the clothing by comparing this value with the predefined reference colors on the card. The comparison was done by calculating the color distance between the fabric's color and the reference colors using the CIEDE2000 color difference formula. The reference color with the smallest distance to the representative color is selected as the identified color of the clothing.

### **Shortcomings of the original approach**

The original implementation was able to identify the color of clothing items, but it required more time than was practical for a competitive setting. The Cybathlon competition involves 10 tasks that must be completed within 8 minutes, meaning each task has an average of about 48 seconds available. The time demands of the original approach made it impossible to complete the task within the given time constraints.

Additionally, the original method processed each clothing item one at a time, which is not effective for detecting the colors of multiple items simultaneously. As the number of items increased, the time required scaled linearly, making it increasingly impractical. To provide a feasible solution for Cybathlon and a useful tool for everyday life it is crucial to reduce the required time for color identification. Therefore, I introduce an approach that is efficient and fast for identifying the color of multiple clothes simultaneously.

# Chapter 3

## Methods

In this chapter I am going to present my solution for the *Colors* task. The methods will be explained in logical order based on the framework of my program showed in Fig 3.1.

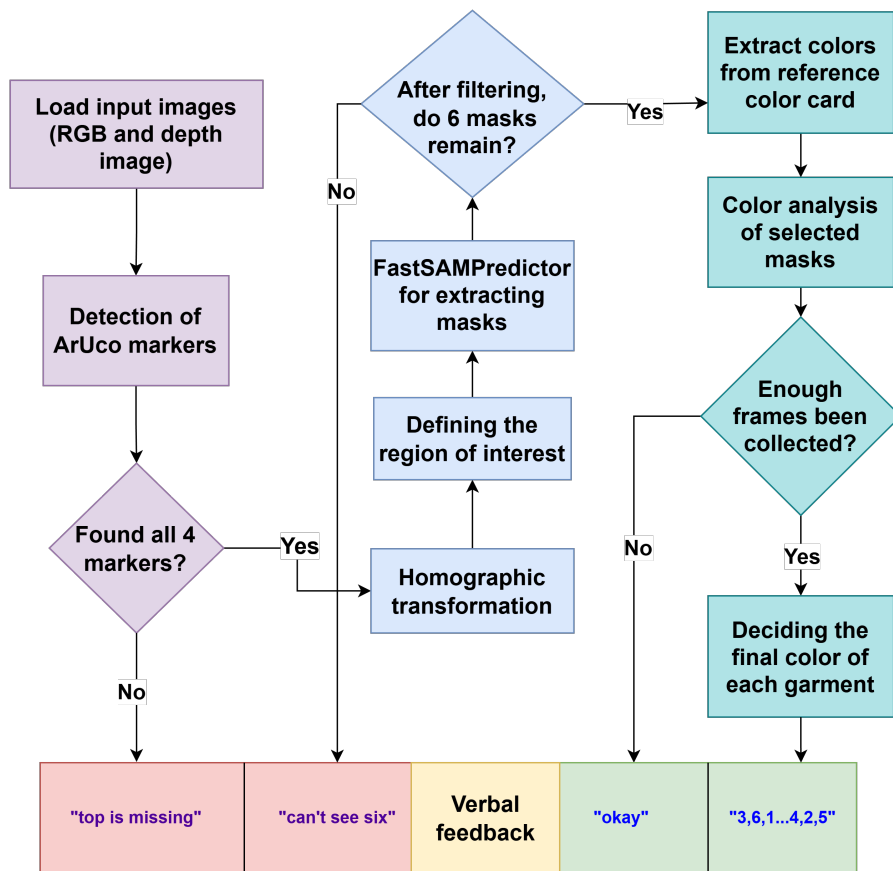


Figure 3.1: This figure shows the framework of the program. Example outputs are color-coded: green indicates successful, while red represents unsuccessful or incomplete frame processing.

### 3.1. Defining the region of interest

The method starts with positioning the reference card, which contains the pre-defined colors, on the clothing items so that all the colors appear along its longer side, perpendicular to the edge. This helps with localization and helps make the effects of the lighting conditions as similar as possible. An example of this is shown in Fig. 3.2.



Figure 3.2: The placement of the card on the clothes.

#### Input images

To provide the foundational data needed to identify, match, and analyze clothing colors, we use two types of input images: firstly, the RGB picture, and secondly, a depth image from the phone’s LiDAR sensor, represented as a single-channel grayscale image.

In this task, we process data from the phone’s LiDAR sensor by limiting the values to a maximum of 2 meters. This means we focus only on distances up to 2 meters, as the region of interest lies well within this range, even with some margin. This approach suits the requirements of the task. The 0 to 2-meter range is represented with sufficient detail using an 8-bit single channel, resulting in a grayscale image. To represent these values, the framework first sets all values greater than 2 to 2, then normalizes the values by dividing by 2 to create a range from 0 to 1, and multiplies by 255 to convert to an 8-bit scale. To recover the actual distances from the processed grayscale images, we need to reverse this transformation. This involves dividing each grayscale pixel value by 255 and then multiplying by 2 to obtain the real depth values. After extracting the depth data, the depth image must be resized to match the dimensions of the corresponding RGB image.

To identify areas of interest, a threshold is applied to the depth image to create a mask of close pixels. Specifically, pixels with depth values less than or equal to a threshold of 0.5 meters are considered *close* and are included in the mask. This mask is used to retain

only the regions of the image that are within a certain proximity, such as the color card and clothing items, while filtering out the background by making those pixels black as demonstrated on Fig 3.3. This ensures that the background is not mistakenly identified as a clothing item.



Figure 3.3: Background filtered based on LiDAR data

### Removing reflections

Reflections can negatively impact the quality of color analysis by distorting the true colors of the clothing items. To address this, a reflection removal process is applied to the input images. The RGB image is first converted to the HSV color space, where reflections are primarily present in the V (value) channel. Adaptive histogram equalization (CLAHE) is then applied to the V channel to enhance contrast and reduce the impact of reflections. By equalizing the brightness levels, the reflections are minimized, and the resulting image more accurately represents the true colors of the clothing items. The modified V channel is merged back with the H (hue) and S (saturation) channels, and the image is converted back to RGB. The result can be seen on Fig 3.4.

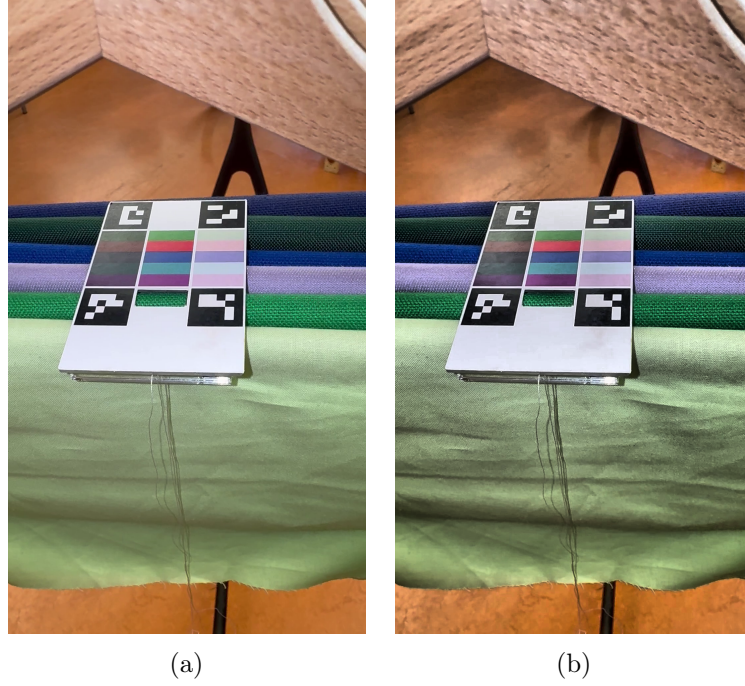


Figure 3.4: A frame (a) before and (b) after CLAHE is applied

### Detection of ArUco markers

The LetSeeApp initially utilized OpenCV’s ArUco library version 4.5.1 for marker detection. However, it became apparent that this version struggled to reliably detect markers in certain scenarios, particularly when images were slightly blurred. As a result, the library was upgraded to version 4.10.0, which introduced significant improvements in detection robustness. This newer version proved crucial in overcoming the limitations of its predecessor, offering enhanced accuracy and performance, even in cases where the images were not perfectly clear.

The size of the markers on the color card was increased to enhance their distinguishability, as smaller ArUco markers in the original approach were more prone to reflections and blurring. This modification was straightforward to implement, as the marker dimensions do not affect the extraction of reference color values from the card. The new color reference card with increased ArUco markers can be seen on figure 3.5.



Figure 3.5: Color reference card

As a reference for future processes, specific coordinates were extracted from the image using OpenCV’s `detectMarkers` [21] function, leveraging the `corners` parameter. From this, four key points, referred to as the *outer corner points*, were identified. These points will be employed in the subsequent homographic transformation. Additionally, four other points, referred to as the *inner corner points*, were extracted. These inner points will play a critical role in the extraction of reference colors.

The positions of both the outer and inner corner points are illustrated on the color card shown in Figure 3.6.

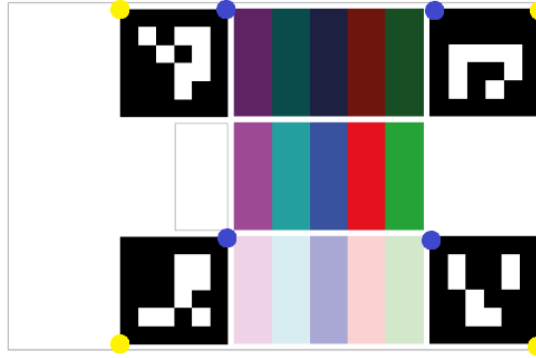


Figure 3.6: Visualization of detected corner points on the reference color card. Outer corner points are marked with yellow circles, while inner corner points are marked with blue circles.

The homographic transformation was performed in a manner similar to the original approach. It maps the original perspective of the color reference card to a rectified, flat perspective, effectively eliminating distortion caused by camera angles. This process ensures that the card appears as if viewed from directly above. The transformation utilizes the four *outer corner points* of the card, mapping them to a new rectangular area that represents the ideal, undistorted view. Additionally, a border is added to include

both the card and the surrounding context, resulting in a rectified image suitable for further analysis.

### Defining the region of interest

To accurately extract the relevant color data from the rectified image, it was necessary to define a specific region of interest (ROI) using the outer corner points of the color card, particularly focusing on the bottom-left and bottom-right points. This method is illustrated in Fig. 3.7.

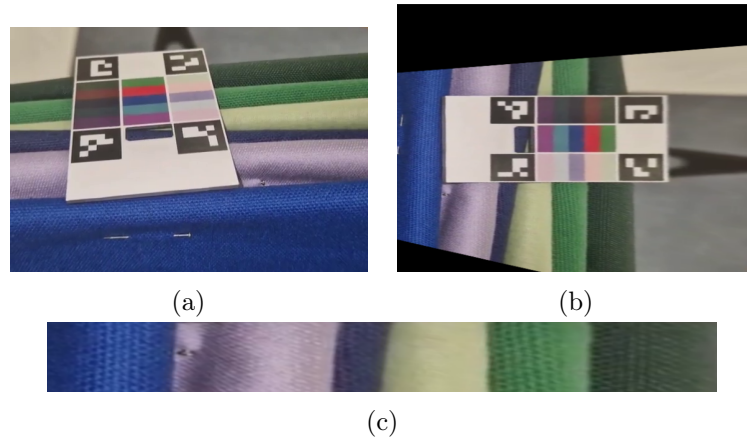


Figure 3.7: Overview of (a) card placement, (b) the output of the homography transformation, and (c) the extracted region of interest

The horizontal boundaries of the ROI are determined using the x-coordinates of these bottom-left and bottom-right points, which define the long edge of the color card. Due to the specific positioning of the card on the clothes, the card is placed near the far end of the clothes, which are longer than the card itself. To account for this, an additional 60% of the distance between these two points is used to shift the edge of the region further to the left, beyond the bottom-left point. This ensures the ROI includes not only the card but also the surrounding fabric.

On the right side, 10% of the distance between the bottom-left and bottom-right points is subtracted from the x-coordinate of the bottom-right point. This adjustment is necessary because the card usually extends slightly beyond the defined corner points, and limiting the region by this small margin prevents unnecessary parts of the card or the background from being included in the ROI.

Vertically, the precision of the boundary is less critical because the width of the region has a minimal effect on the color detection process. Therefore, a simple shift is applied to

the y-coordinate, starting slightly below the bottom-left corner and extending upwards by a fixed pixel distance.

The defined region of interest captures all essential areas of the image that contain the items whose colors will be analyzed in subsequent stages of the analysis.

### 3.2. Clothing segmentation and mask extraction

Following the extraction of the region of interest, the next step is to segment the six clothing items and extract their corresponding masks. To accomplish this, I experimented with various methods.

#### Tried and tested methods

Initially I used the OpenCV implementation [22] of the **K-Means clustering** algorithm to segment the six clothing items in the image. K-Means clustering groups similar colors into distinct clusters, and while it can identify the six dominant colors of the clothing items, it does not consider spatial context. This limitation resulted in inconsistent segmentation of individual garments as it can be seen on Fig 3.8.



Figure 3.8: K-Means clustering on ROI

To improve K-Means clustering for images, I experimented with several preprocessing methods, consistently beginning with Gaussian blur to reduce noise. **SLIC superpixel segmentation** was considered firstly, as it can maintain spatial coherence. By segmenting the image into compact superpixels, I aimed to enhance clustering accuracy by preserving spatial context. Each superpixel's average color was then calculated, creating a simplified color palette that captures the image's main color regions while mitigating random noise. With the superpixels prepared, I applied K-Means clustering to the averaged superpixel colors rather than individual pixels, setting a target of six color clusters. Finally, I mapped each cluster center back to its corresponding superpixel region in the original image. Although segmentation improved, shadows in the image still introduced inconsistencies that reduced clustering quality. The result is demonstrated on Fig 3.9.



Figure 3.9: SLIC superpixel segmentation before K-Means

Following this, I used **mean shift** with the aim of smoothing the image and emphasizing areas of high pixel density, providing a cleaner and better-segmented input to K-Means. While this smoothing effect did reduce some noise, it could not completely eliminate the effect of shadows. The result is presented on Fig 3.10.

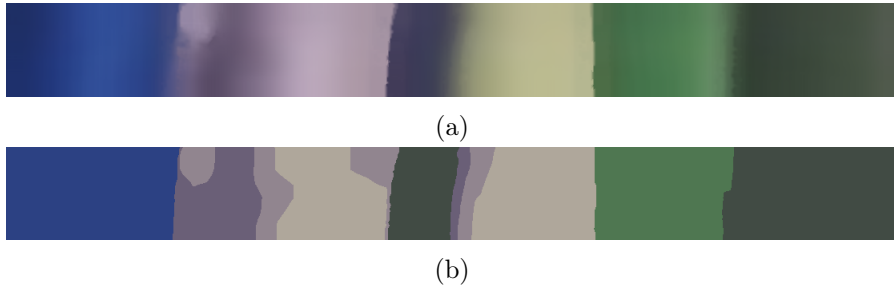


Figure 3.10: (a) ROI after mean shift and (b) then K-Means applied

Another approach explored was contour detection. Since the clothes were hanging, there were deep shadows along the lines where they touched. I saw these shadows as potentially useful for clearly delineating individual garments. Initially, I experimented with **adaptive thresholding**, hoping it could achieve accurate edge detection, which could aid in distinguishing each piece of clothing. However, adaptive thresholding struggled in practice—some edges were missed or inaccurately detected, as illustrated in Fig 3.11.

To improve on this, I turned to **Otsu's thresholding**, which can automatically determine an optimal threshold value for separating the foreground from the background by minimizing intra-class variance. I expected that Otsu's method would provide a clearer binary separation, enhancing segmentation by using the deep shadows between the garments as a natural boundary. This method produced better results than adaptive thresholding, as it successfully captured more of the garment contours. However, not all contours were consistently detected, and occasionally the method picked up extra edges or missed finer details, leading to some incomplete segmentation. The result is illustrated in Fig 3.12.

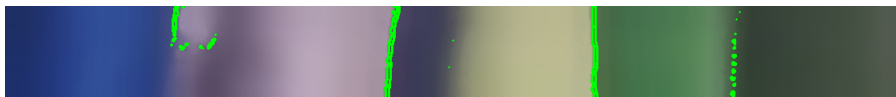


Figure 3.11: The effect of adaptive thresholding



Figure 3.12: The effect of Otsu's thresholding

### Chosen method: FastSAMPredictor

After experimenting with several traditional image processing techniques, we decided to use the pre-trained FastSAMPredictor neural network for segmenting the clothing items. Neural networks were initially avoided due to concerns about their inherent complexity, the significant computational resources they demanded, and the extensive amount of annotated data required. These factors were expected to be time- and resource-intensive, especially given the limited data available, which increased the risk of overfitting and hindered the model's capacity to generalize effectively. However, the inability of these conventional methods to consistently and accurately segment the clothing items made it clear that a more advanced solution was required.

The FastSAMPredictor did not add as much computational time as initially expected, while providing a significant improvement in segmentation accuracy and yielding much more reliable results compared to the earlier techniques. The neural network generated multiple high-quality masks. Many of the returned masks overlapped each other as it can be seen on Fig 3.13. The next step is to review the generated masks to identify the six relevant masks that correspond to the six clothing items in the image.



Figure 3.13: Masks returned by FastSAMPredictor

### Postprocess of the extracted masks

Refining the extracted masks is necessary because the prediction model often produces many irrelevant masks. To ensure accurate color matching and object detection, the top six masks most likely to represent the clothing items are selected, while others are discarded. The post-processing algorithm includes steps like resizing, filtering based on area and overlap, and selecting regions based on size and spatial consistency. This process effectively reduces noise and isolates key regions, ensuring further analysis focuses only on meaningful data.

The first step in processing each mask involves resizing it to match the dimensions of the predefined Region of Interest (ROI). This step is crucial because FastSAM adjusts all input images to a standard size, resulting in output masks that differ from the original image dimensions. The model performs resizing to ensure that all inputs are consistent, allowing the convolutional layers and the network to operate as intended without any issues related to incompatible sizes [13].

To enhance accuracy, the algorithm evaluates the overlap between the resized masks and a black background mask. A black background mask is generated by identifying pixels in the region of interest (ROI) that are completely black. This mask highlights regions where LiDAR has filtered out the background, which are irrelevant to the Region of Interest and must be excluded from further analysis. By quantifying the overlap, the algorithm can distinguish detected objects from irrelevant areas effectively. The overlap ratio measures the number of overlapping pixels between the mask and the background, providing an objective metric of how much of the mask intersects with the undesired background. Masks exhibiting more than 20% overlap with the background are discarded. This threshold ensures that only masks representing valid objects of interest are retained for further analysis, as significant overlap indicates the mask primarily represents the background rather than a meaningful object.

The next step involves calculating the area of each mask and performing overlap filtering to ensure only valid regions are retained. The area of a mask is determined by counting the number of non-zero pixels, which represent the detected region. This provides a quantitative measure of the size of each detected object. Masks with an area smaller than a threshold value, determined experimentally to be 3% of the overall number of pixels, are filtered out. This step is necessary because smaller masks are likely to be covered by normal-sized masks, which would result in the larger, valid mask being discarded during overlap filtering. By removing small-area masks first, we improve the chances of retaining valid masks.

After calculating the area, further overlap filtering is conducted to eliminate redundant or irrelevant regions. Masks that cover an excessive portion of another mask are discarded, as these are likely to be either duplicates or noise. Specifically, if one mask overlaps with 90% or more of another mask, the smaller mask is retained, and the larger one is removed. This threshold is set to 90% because, although there may still be smaller or invalid masks present, we do not want the valid masks to be discarded due to minor overlaps. Overlap filtering is particularly necessary because there are often large masks

that cover multiple items of clothing, but we cannot simply filter based on a large area, as clothing masks can sometimes be quite large depending on the camera angle.

For each mask, the centroid is computed using image moments (specifically using  $m_{00}$ ,  $m_{01}$ , and  $m_{10}$  as described in Section 2.2), which provides a representation of the geometric center of each region. The centroid's coordinates  $(c_x, c_y)$  are derived as:

$$c_x = \frac{m_{10}}{m_{00}}, \quad c_y = \frac{m_{01}}{m_{00}}$$

where  $m_{00}$  represents the total area of the mask, and  $m_{10}$  and  $m_{01}$  are the first-order moments along the x and y axes, respectively.

After calculating the centroids, a RANSAC regression-based filtering step is applied to further refine the set of candidate masks. This filtering is crucial because not all detected masks correspond to the clothing items of interest. The clothing items are expected to be spatially adjacent, and any mask with a centroid significantly distant from the others is unlikely to represent a valid clothing region. Instead, such masks might represent parts of the person's body, such as a hand or finger, or even background elements. These incorrect masks are identified by their centroids being spatially isolated from the cluster of valid centroids. RANSAC regression is used to distinguish between centroids that are spatially coherent (inliers) and those that are not (outliers). After RANSAC filtering, a refined set of masks with spatially coherent centroids is obtained as shown in Fig 3.14.

Finally, the valid masks are sorted by area in descending order, and the six largest masks are selected to ensure all expected clothing items are represented.

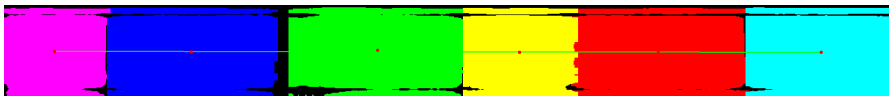


Figure 3.14: The selected 6 mask with a mark on their centroids and the line of RANSAC regression

### 3.3. Color analysis and auditory feedback

#### 3.3.1 Color analysis

##### Reference color determination

Before determining the colors of the segmented masks we need to extract the colors from the reference color card. These colors serve as a basis for comparing sampled colors and determining the closest match.

The reference colors are determined from the color card that contains 3 rows and 5 columns. The 5 columns represent the main colors: pink, cyan, blue, red, and green, while the 3 rows represent their respective shades: dark, mid, and light. To extract the reference colors, the positions of both inner and outer corner points of the color card are detected using ArUco markers, as described in section 3.1. These corner points help define a rectified grid, allowing for the precise localization of each color swatch on the card. The grid is created by first calculating the distance along the x-axis between the two inner top corner points, which represent the width of the color swatches. This distance is then divided by the number of main colors (5) to determine the step size along the x-axis for each column. Similarly, the distance along the y-axis between the top inner corner and the bottom outer corner is calculated and divided by the number of shades (3) to determine the step size along the y-axis for each row. Using these step sizes, the card is effectively divided into 3 rows and 5 columns, creating a grid that precisely defines the location of each color swatch. This grid provides clear boundaries for each color swatch, ensuring that sampling is performed within the correct regions.

For each color swatch, three sampling points are taken: one at the center of the swatch and two at offset rows above and below the center. The location of the sampling points are demonstrated on Fig 3.15. The x-coordinate for sampling is given by:

$$x = x_{\text{top\_left}} + (i + 0.5) \times \text{steps}_{\text{on-x-axis}}$$

where  $i$  is the column index of the swatch. The y-coordinate for the base row is given by:

$$y_{\text{base}} = y_{\text{top\_left}} + (j + 0.5) \times \text{steps}_{\text{on-y-axis}}$$

where  $j$  is the row index. The two offset rows are determined by adding and subtracting an offset value from the base row:

$$y_{\text{offset\_1}} = y_{\text{base}} - \frac{\text{steps}_{\text{on-y-axis}}}{3}$$

$$y_{\text{offset\_2}} = y_{\text{base}} + \frac{\text{steps}_{\text{on-y-axis}}}{3}$$

By averaging the RGB values from these three sampling points, the algorithm derives a representative color value for each swatch. This approach ensures a higher degree of accuracy compared to using a single point, as it helps smooth out any inconsistencies

that might exist in the color swatch due to uneven lighting or imperfections in the card's surface. The final output is a set of reference colors, each represented by a name (e.g., "dark pink," "mid cyan") and an average RGB value.

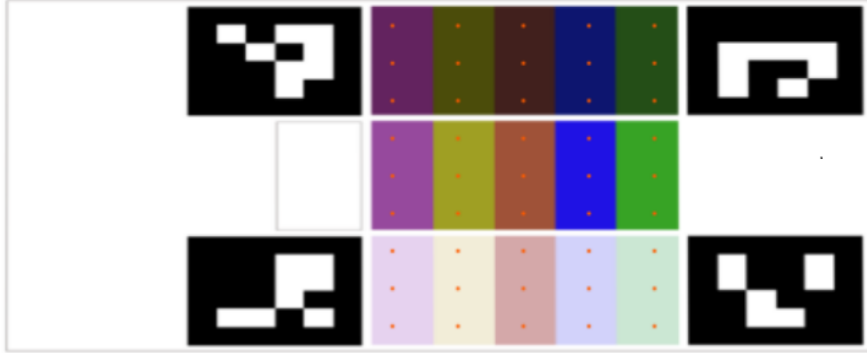


Figure 3.15: Colors are sampled from the reference color card

### Color analysis of selected masks

The first step in the color analysis process of the clothing items is to sort the selected masks based on the x-coordinates of their centroids. Sorting the masks from left to right is essential because it allows us to consistently identify and analyze each clothing item in the correct order. Knowing the order of the clothing items helps us ensure that the color analysis is performed systematically, avoiding confusion or errors in identifying which mask corresponds to which item.

After sorting the masks, it is crucial to remove any overlapping regions between them. Overlapping areas occur when a pixel is part of more than one mask, but in reality, it can only belong to a single clothing item. To avoid mistakes in color sampling, overlapping regions are identified and filtered out from the masks involved. This ensures that each mask exclusively represents a unique item of clothing. By removing overlaps, we avoid cross-contamination of colors between adjacent clothing items, resulting in a more accurate determination of each item's true color.

To determine the color of each clothing item accurately, multiple sample points are required. Using only one sampling point per clothing item is insufficient because it cannot capture the variability of color across the entire item. Therefore, 20 sample points were used for each mask to ensure a more comprehensive representation of the item's color.

The sample points are selected based on a uniform distribution across the target region of each mask, as presented on Fig 3.16. Uniform distribution ensures that the sampling points are spread evenly throughout the selected area, providing a well-rounded representation of the overall color. Specifically, the selected sample points are primarily

taken from the upper part of each clothing item, which appears as the right side of the region of interest (ROI) in our setup. This area is chosen because it is less likely to be covered by shadows, resulting in more accurate color sampling.

The sampling process involves dividing the target region into a grid and selecting points within each grid cell. This grid-based approach ensures that all areas within the region are sampled, avoiding any bias towards a particular part of the clothing item. For the first mask, special handling is applied because it often has a bright line of light running through the middle. This bright line could skew the color analysis if sample points were taken from it. To address this, sample points are selected from the left and right sides of the centroid, avoiding the bright central region.

The consistency in sampling is further improved by knowing where shadows are likely to occur. This is possible because the flashlight on the phone is always turned on, and our pilot examines the clothing items from the same direction each time. This controlled environment allows us to predict shadow placement and avoid those areas during sample point selection, resulting in a more accurate determination of the colors of the clothing items.

After selecting the sample points, the next step is to analyze the colors of these points to determine the most representative color for each clothing item. To do this, we calculate the distance between the sampled color values and the reference colors. The color distance is computed using the CIEDE2000 color difference formula to determine the difference from each of the reference colors.

By calculating the color distance for each sample point, we can identify the reference color that is closest to the sampled value. This process is repeated for all sample points within a mask, allowing us to gather multiple reference color matches. The goal is to find the color that is most consistently represented across the sampled points, which will serve as the most representative color for the entire clothing item.

While determining the representative color for the clothing item, we also store the distance of each sample point from all reference colors as this data will be useful in a subsequent process.

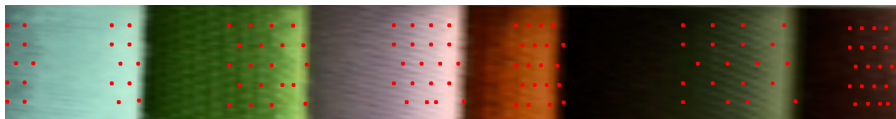


Figure 3.16: Colors are sampled from the clothing items

## Deciding the final color of each garment

I introduced a memory function in the program where we can set a number that determines the number of frames that are accumulated before making a final decision. Accumulating multiple frames helps avoid issues such as retaining incorrect masks in a single frame or poor segmentation of the masks. By analyzing a series of frames, potential errors are minimized, resulting in a more reliable and accurate determination of the colors.

In the first step we analyze the colors of the six clothing items to identify the dominant color for each dress. The colors examined at this stage are the most representative colors of each dress in the given frame. The decision is made by selecting the most common color. As described earlier, these representative colors are determined by analyzing the sampled points, and selecting that particular color the majority of sampling points were closest to.

The next step is to analyze the prevalent color families among the six clothing items. This step is necessary because each color (a combination of shade and main color) must be unique across all six items. Moreover, we know that the six clothing items are composed of three shades each of two main colors. Therefore, it is crucial to determine which two main colors are present to effectively handle identical colors and assign the appropriate variations. To achieve this, the colors of each dress are split into their shade and main color components (e.g., “light blue” is split into “light” and “blue”). Then, the two most common color families are identified across all the dresses.

The next challenge is ensuring that all required variations are present and that no identical colors are assigned to multiple dresses. Since the six clothing items are composed of three shades of two main colors, it is essential to have light, medium, and dark shades of each main color. Any missing variations must be assigned appropriately to maintain this balance, and any identical colors must be adjusted to ensure uniqueness.

The process starts by creating a list of the required variations for each color family. These include the “light,” “mid,” and “dark” shades for both common color families identified in the previous steps. The algorithm then compares these required variations to the colors currently assigned to each dress. Any missing color variations are recorded for further processing.

The algorithm then identifies any dresses that have identical colors or that have a main color different from the two most common color families found earlier. These dresses are collected into a list for reassignment. A distance matrix is constructed, where each element represents the average distance between a collected color variation and a

dress needing reassignment. The average distance is determined by first calculating the mean of the distances from each of the 20 color samples to the target color, followed by averaging these values across all accumulated frames to ensure the most accurate color match.

To find the optimal assignment of colors, the Hungarian algorithm is applied to the distance matrix. This algorithm minimizes the total distance between the dresses and the collected colors, providing the most accurate assignment. Once the optimal matches are determined, the colors are updated accordingly, and each dress receives a unique and appropriate color variation, ensuring that all required shades are represented and no two dresses share the same color.

### **3.3.2 Auditory feedback**

#### **Missing ArUco markers**

After detecting ArUco markers, as described in Section 3.1, auditory feedback is provided if less than four markers are detected. Detecting all four markers is crucial for subsequent steps. The feedback specifies which markers are missing, such as “Bottom left is missing” or “Top side is missing,” allowing the user to adjust the camera positioning to ensure all markers are detected, thereby improving system accuracy.

#### **Mask detection for clothing items**

As explained in Section 3.2, the process involves filtering masks for six clothing items. If fewer than six masks are detected, the system gives auditory feedback saying “Can’t see six.” This means there was an issue during segmentation, even though all ArUco markers were properly detected, and further adjustments are needed to fix the problem.

#### **Data accumulation status**

If the process is going well but more frames are still needed for data accumulation, the system provides the auditory feedback “OK.” This indicates that no adjustments are necessary, but more data is being collected to enhance accuracy. This helps the user remain patient during the accumulation phase without making unnecessary changes.

#### **Color analysis results**

After enough frames have been accumulated, the algorithm described in Section 3.3.1 determines the colors of the six clothing items. The system assigns a serial number to

each item by analyzing its main color and shade. Each shade is categorized as light (0), mid (1), or dark (2), and each unique color is assigned a value, with the first color being 1 and the second being 4. The serial number is calculated by summing these shade and color values, resulting in a distinct identifier for each item.

The calculated serial number is then communicated to the pilot, who uses it to arrange the items in the correct order. The pause in the middle of the verbal feedback helps the pilot process what he heard, reducing cognitive strain and organizational errors.

### 3.4. Implementation frameworks

I initially implemented the prototype in Python because of its simplicity and flexibility. Python allowed me to quickly prototype the system, making it easier to iterate on ideas and test different approaches. The core functionality was organized as a function-calling pipeline, with each function representing a distinct step in the process.

However, due to performance constraints and the requirement for real-time processing on a mobile device, I translated the final implementation to C++ to achieve greater speed and efficiency. The main objective was to have the program run independently on the mobile device, without relying on streaming data to an external computer, which would have allowed continued use of Python but required more powerful hardware. The solution had to be integrated into the existing Swift and C++ framework on the mobile device, making a native implementation necessary. To facilitate integration with the mobile app, I used an object-oriented approach (OOP) in C++. A dedicated class with two primary functions was created to interface with the Swift code. The first function processes the input RGB and grayscale images to extract regions of interest (ROI). Since the FastSamplePredictor model could not be directly initialized in C++, it was handled in Swift. Swift initializes FastSamplePredictor and then calls the second function in the C++ class, passing the processed masks for further analysis.

## Chapter 4

# Results

In this chapter, I present an evaluation of the performance of my program, based on tests conducted using four 30-second videos captured with an iPhone 15 Pro. This choice mirrors the equipment used during the competition, ensuring consistency between the test environment and the actual competition scenario. Each video contained different combinations of clothing colors, filmed in a room with a mix of artificial and natural lighting, with the phone’s flash turned on to replicate the competition conditions. The videos were recorded at 30 frames per second, but only every 8th frame was processed, aligning with the fps settings of the LetSeeApp to maintain compatibility and similar performance metrics.

To assess the algorithm’s performance, I performed three sets of tests for each video, adjusting the “memory” parameter, which represents the number of frames accumulated before making a decision (as described in Chapter 3.3.1). The tests used memory settings of 1, 3, and 6 to evaluate how memory impacts accuracy and decision-making. The ArUco markers were successfully detected in 96.35% of frames, mainly due to the updated version of the OpenCV library (v4.10.0), which offered enhanced marker detection capabilities compared to previous versions. Another critical step in the program was finding the masks for all six clothing items, which occurred with an 88.89% success rate. These high detection rates were crucial for ensuring that the subsequent color detection steps could proceed.

I evaluated the accuracy of finding the color of all six clothing items according to the competition rules. This means that there can be only two domain colors, each with three associated shades. This accuracy varied between 87.75% and 93.4%, depending on the memory setting (as shown in Table 4.1). I also measured the accuracy of finding the right color for individual clothing items. This yielded better results, with accuracy

in the 92-95% range. During the competition, however, it was sufficient for the pilot to know the correct order of the clothes, regardless of exact color identification. The program sometimes confused similar colors, such as pink and red, which contributed to lower color detection accuracy compared to the order detection accuracy which was in the 92-95% range. This was particularly evident in scenarios where lighting conditions introduced subtle variations in color perception, further complicating accurate differentiation between similar hues.

I also measured how frequently the Hungarian algorithm was needed. As described in Chapter 3.3.1, this algorithm, while effective, is computationally expensive and is employed only when there is no unambiguous color for a clothing item, ensuring it is used minimally and only when necessary. The results, presented in Table 4.2, show that the Hungarian algorithm was required the majority of the time, making it a crucial component for achieving accurate color and order detection. Accumulating frames before making decisions improved the overall accuracy but posed a risk if the pilot moved and missed the required number of frames. The time taken for the program to process from the first frame until it announced the final word of the color results is presented in Table 4.3, comparing various memory settings. These times are only valid if there were no issues in frame detection, ensuring that the processed frames followed each other smoothly without delays. During the competition, we opted for a memory setting of 3, which offered a good balance of speed and accuracy, allowing the pilot to receive timely feedback while maintaining an acceptable level of detection reliability.

| <b>Metric</b>                                        | <b>M<sub>1</sub></b> | <b>M<sub>3</sub></b> | <b>M<sub>6</sub></b> |
|------------------------------------------------------|----------------------|----------------------|----------------------|
| Accuracy of color detection for all six garments (%) | 87.75                | 91.67                | 93.4                 |
| Accuracy of color detection (%)                      | 95.96                | 96.72                | 97.12                |
| Accuracy of garment order detection (%)              | 92.25                | 94.67                | 95.06                |

Table 4.1: Accuracy metrics in cases where colors were reported *Note: Metrics represent the overall performance under conditions where the program successfully reported colors. This includes cases where both ArUco markers and clothing masks were detected. The columns reflect performance across memory configurations of 1, 3, and 6 frames.*

Another important consideration during the competition was the stability of the color reference card. We used a card with a plastic hook that allowed the pilot to attach it to the clothing, ensuring it remained stable. This eliminated the need for LiDAR filtering, which would otherwise be necessary to account for card movement. However, for more general scenarios where the card may not have this support, background interference can degrade performance. I conducted additional tests where the reference card was placed such that the background was within the region of interest (ROI). In these tests, enabling

| <b>Metric</b>                                             | <b>M<sub>1</sub></b> | <b>M<sub>3</sub></b> | <b>M<sub>6</sub></b> |
|-----------------------------------------------------------|----------------------|----------------------|----------------------|
| Relative frequency of calling the Hungarian algorithm (%) | 61.33                | 65.15                | 70.37                |
| Accuracy of color detection for all six garments (%)      | 82.25                | 87.2                 | 84.21                |
| Accuracy of garment order detection (%)                   | 88.77                | 91.86                | 92.98                |

Table 4.2: Performance metrics in cases where colors were reported and Hungarian algorithm applied. *Note: Metrics represent the overall performance under conditions where the program successfully reported colors. This includes cases where both ArUco markers and clothing masks were detected and the Hungarian algorithm was required. The columns reflect performance across memory configurations of 1, 3, and 6 frames.*

|          | $M_1$ | $M_3$ | $M_6$ |
|----------|-------|-------|-------|
| Time (s) | 3.0   | 4.5   | 7.0   |

Table 4.3: Processing time for different memory settings

LiDAR filtering significantly improved the performance, increasing color accuracy by 30% and order accuracy by 40%. This demonstrated that LiDAR filtering can be a valuable addition in less controlled environments, providing robustness against background noise and improving the overall reliability of the color detection process. While these accuracy metrics do not guarantee perfect results, the high frame rate of the camera allows the pilot to receive repeated updates, improving reliability over time. This repeated feedback mechanism compensates for occasional inaccuracies by providing multiple opportunities for correction.

One of the primary goals at the beginning of this work was to improve the previous implementation of the color task in terms of time efficiency. To evaluate the effectiveness of the improvements, I conducted a test involving a participant with no visual impairments. The participant completed five rounds of the task, with her eyes covered, using both the previous and the improved versions of the software using the same settings as in the competition. I measured the time from the moment the card touched the clothes until all garments were sorted. The results demonstrate that the participant was able to complete the task in an average of 107 seconds with the previous version and 33 seconds with the presented, improved version. This significant reduction in task completion time indicates that the initial goal of enhancing time efficiency has been successfully achieved. Moreover, the new version not only outperforms the previous one but also meets the one-minute completion goal set at the beginning of this project. During the competition, we were satisfied with these results, as we completed the entire task in 1 minute and 24 seconds, including walking to the clothing, locating it, and accessing the color program on the phone. The core task of ordering the clothes using the program, from placing the

color cards to finishing, took only 38 seconds, demonstrating the efficiency of the system under stressful conditions.

## Chapter 5

# Conclusion and future work

### 5.1. Project summary

This project provides an innovative a solution for identifying mutiple colors simoul-  
taneuously in the *Colors* task of Cybathlon, and it was successful as the EyeRider team  
achieved first place by completing 7 out of 10 tasks, including the *Colors* task. The  
main objective was to create an efficient system for identifying and sorting clothing col-  
ors to assist visually impaired individuals in a competitive setting, which was successfully  
demonstrated during the event.

A major challenge in the project was finding a reliable tool for clothing segmentation  
that could handle varying lighting and shadows. Traditional image processing methods  
struggled under these conditions, making consistent segmentation difficult. To address  
this, I utilized the Fast Segment Anything Model (FastSAM), which greatly improved  
segmentation accuracy. Shadows and lighting variations were further managed using  
Contrast Limited Adaptive Histogram Equalization (CLAHE) and a strategic sampling  
approach, with multiple points selected from each clothing item while avoiding highlighted  
areas for better color accuracy.

Another challenge was developing a program to determine the best color for each  
clothing item, while also considering the colors assigned to the other items. The Hun-  
garian algorithm was employed when the color classification was ambiguous, aiming to  
optimize the assignment so that each item receives a unique and correct classification.  
This approach led to high accuracy rates, with garment order detection accuracy at  
92.98% and color detection accuracy at 97.12%, demonstrating effectiveness in compet-  
itive conditions. The new implementation also significantly improved time efficiency,

reducing the average completion time of the *Colors* task by 69%, from 107 seconds to just 33 seconds, thereby meeting the project’s goals of increased speed and reliability.

Beyond the competition, this project contributed to assistive technologies by enhancing the independence of visually impaired individuals. Although initially developed for Cybathlon, its capability to determine multiple object colors simultaneously with high accuracy holds potential for real-time problem-solving applications.

## 5.2. Discussion and future work

The solution developed for this task has undergone further development, primarily to address the issues posed by poor lighting conditions that became evident during the competition, where the lighting conditions were often unpredictable, with strong artificial light coming from reflectors at different angles.

In response to the lighting issues, additional adjustments were made to ensure consistent detection under suboptimal conditions. I proposed repositioning the plastic attachment to the fifth garment instead of the sixth, allowing the pilot to fold up the last garment to create a “tent” that minimized reflections from strong lights while enhancing illumination from the phone’s flash. This simple adjustment provided a makeshift solution to control lighting, reducing harsh reflections and shadows. Another improvement was replacing the color patches on the color reference card with samples from the actual garment materials, rather than using printed color patches. This change made them more resistant to reflections and better replicate the effect of light on real garments, achieving more accurate color representation, especially under mixed lighting.

In many cases, the reflection from the light on the color reference card made it challenging to detect all four ArUco markers, necessitating the ability to achieve accurate results using any combination of detected markers. This reflection issue was particularly problematic when strong directional lights caused glare on the card’s surface, obscuring key markers. Initially, we started with the idea of using four reference points for the homographic transformation, but determining which four points to use proved difficult when not all markers were available. Therefore, the method presented in this paper was extended to enable homography transformation upon detecting any ArUco markers, even in the case of a single marker. My suggestion was that all four corner points of each detected ArUco marker could be used for robustness. However, the OpenCV function interface allows only 4 points, so in the end, we calculate the most robust solution based

on 4 points determined by the detected markers. The updated approach was further refined by my supervisors.

For future work, there are several areas in which this implementation could be further enhanced. Currently, the dominant color of each garment is determined by selecting the most commonly identified representative color across frames. While this method worked adequately, and the Hungarian algorithm helped to correct inaccuracies by reassessing color matches, there is room for improvement. An enhancement could be made by incorporating color distances into the decision-making process when determining the dominant color. Furthermore, using squared distances could also be considered for the distance matrix of the Hungarian matching to increase the penalty for larger mismatches more than smaller ones. These would help refine the accuracy of color identification, especially in scenarios where subtle differences are crucial. In addition, incorporating machine learning models trained on various lighting conditions could enhance the robustness of the detection mechanism.

While this implementation is effective for the specialized *Colors* task, it is not yet general enough for broader real-world applications. To adapt it for general use, the program should be capable of handling scenarios where the number of garments or primary colors is unknown in advance. Currently, the number of recognizable colors is adjusted to match the size of the reference card, which limits the number of colors that can be detected. While this design has been optimized for the current use case, it could be further generalized to allow for more flexibility in the number of detectable colors.

Currently, the program outputs numeric identifiers, which is tailored for the competition requirements. However, it could be easily modified to announce the name of the color that most closely matches each garment. In real-world scenarios, it might be acceptable for multiple garments to have the same color, unlike the competition where this was not allowed. Simply disabling the constraint that requires unique colors for each garment would make the system more adaptable for everyday use. Although the current reference card includes numerous basic colors and their shades, exploring whether it can be robustly supplemented with additional colors, such as yellow, would enhance its generalizability. Addressing these future challenges could expand the solution’s impact beyond the competition, serving as a practical tool in everyday applications or assistive technology.

# Bibliography

- [1] *Cyathlon 2024 races and rules*, Accessed: 2024-11-13, ETH Zürich, 2022. [Online]. Available: [https://cyathlon.ethz.ch/documents/races-and-rules/CYBATHLON%5C%202024/CYBATHLON%5C\\_RacesAndRules%5C\\_2024.pdf](https://cyathlon.ethz.ch/documents/races-and-rules/CYBATHLON%5C%202024/CYBATHLON%5C_RacesAndRules%5C_2024.pdf).
- [2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th. Pearson International, 2017. [Online]. Available: <https://elibrary.pearson.de/book/99.150005/9781292223070>.
- [3] G. Sharma and R. Bala, *Digital color imaging handbook*. CRC press, 2017.
- [4] K. Zuiderveld, “Contrast limited adaptive histogram equalization,” in *Graphics gems IV*, Academic Press, 1994, pp. 474–485.
- [5] M.-K. Hu, “Visual pattern recognition by moment invariants,” *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.
- [6] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [7] N. Otsu *et al.*, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.
- [8] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.
- [9] OpenCV, *Understanding k-means clustering*, Accessed: 13 October 2024, n.d. [Online]. Available: [https://docs.opencv.org/4.x/de/d4d/tutorial\\_py\\_kmeans\\_understanding.html](https://docs.opencv.org/4.x/de/d4d/tutorial_py_kmeans_understanding.html).
- [10] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
- [11] X. Ren and J. Malik, “Learning a classification model for segmentation,” *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 10–17, 2003.

- [12] X. Zhao, W. Ding, Y. An, *et al.*, *Fast segment anything*, 2023. arXiv: 2306.12156 [cs.CV].
- [13] G. Jocher, J. Qiu, and A. Chaurasia, *Ultralytics yolo*, <https://github.com/ultralytics/ultralytics>, version 8.0.0, AGPL-3.0 License, Jan. 10, 2023. [Online]. Available: <https://ultralytics.com>.
- [14] M. Gong, D. Wang, X. Zhao, H. Guo, D. Luo, and M. Song, “A review of non-maximum suppression algorithms for deep learning target detection,” in *Seventh Symposium on Novel Photoelectronic Detection Technology and Applications*, J. Su, J. Chu, Q. Yu, and H. Jiang, Eds., International Society for Optics and Photonics, vol. 11763, SPIE, 2021, p. 1176332. DOI: 10.1117/12.2586477. [Online]. Available: <https://doi.org/10.1117/12.2586477>.
- [15] R. Varghese and S. M., “Yolov8: A novel object detection algorithm with enhanced performance and robustness,” in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 2024, pp. 1–6. DOI: 10.1109/ADICS58448.2024.10533619.
- [16] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [17] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [18] H. Péter, *Telefonnal a látássérültekért: Mobilszenzorokra épülő, ai és a gépi látás alapú algoritmusfejlesztés*, Tutored Resarch Project, 2024 Spring, *Not publicly available*.
- [19] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014. DOI: 10.1016/j.patcog.2014.01.005.
- [20] OpenCV, *Detection of aruco markers*, Accessed: 12 October 2024, n.d. [Online]. Available: [https://docs.opencv.org/4.x/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html).
- [21] OpenCV, *Cv::aruco::arucodetector class reference*, Accessed: 20 October 2024, n.d. [Online]. Available: [https://docs.opencv.org/4.x/d2/d1a/classcv\\_1\\_1aruco\\_1\\_1ArucoDetector.html#a0c1d14251bf1cbb06277f49cfe1c9b61](https://docs.opencv.org/4.x/d2/d1a/classcv_1_1aruco_1_1ArucoDetector.html#a0c1d14251bf1cbb06277f49cfe1c9b61).

- [22] OpenCV, *K-means clustering in opencv*, Accessed: 13 October 2024, n.d. [Online]. Available: [https://docs.opencv.org/3.4/d1/d5c/tutorial\\_py\\_kmeans\\_opencv.html](https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html).